Implementación de un juego de estrategia con IA y multijugador (Linja)

Tomás Robalo Moreno

14 de septiembre de 2015

Datos del Proyecto

Título: *Implementación de un juego de estrategia con IA y multijugador (Linja)*

Autor: *Tomás Robalo Moreno*

Titulación: *Ingeniería en Informática de Gestión*

Créditos: *22,5*

Director: *Miquel Barceló Garcia*

Departamento: *ESSI*

Miembros del Tribunal

Presidente: *Xavier Burgués Illa*

Vocal: *Angel Olivé Duran*

Secretario: *Miquel Barceló Garcia*

Calificación

Calificación numérica:

Calificación descriptiva:

Fecha:

Índice

1. Objetivo del proyecto

El objetivo de este proyecto es la adaptación para dispositivos móviles del juego de mesa Linja. Este proyecto comprende todas las fases de desarrollo: el análisis, la especificación, el diseño y la implementación del juego, hasta alcanzar una versión del mismo óptima para su publicación.

Una de las funcionalidades básicas que se ha creído debe ofrecer una adaptación de este tipo es la de permitir que el usuario pueda enfrentarse a diferentes oponentes de Inteligencia Artificial. Así pues, una parte importante de este proyecto se centrará en la creación e implementación de un algoritmo que dé respuesta a la necesidad de disponer de una IA que se desenvuelva con cierta competencia contra el usuario.

Una segunda funcionalidad que también se contempla necesaria es la de permitir que dos usuarios reales puedan enfrentarse en una partida, ya sea en el mismo dispositivo o a través de Internet. Por lo tanto otra parte de este proyecto se centrará en la implementación de un sistema de juego local y online que ofrezca al usuario dichas opciones.

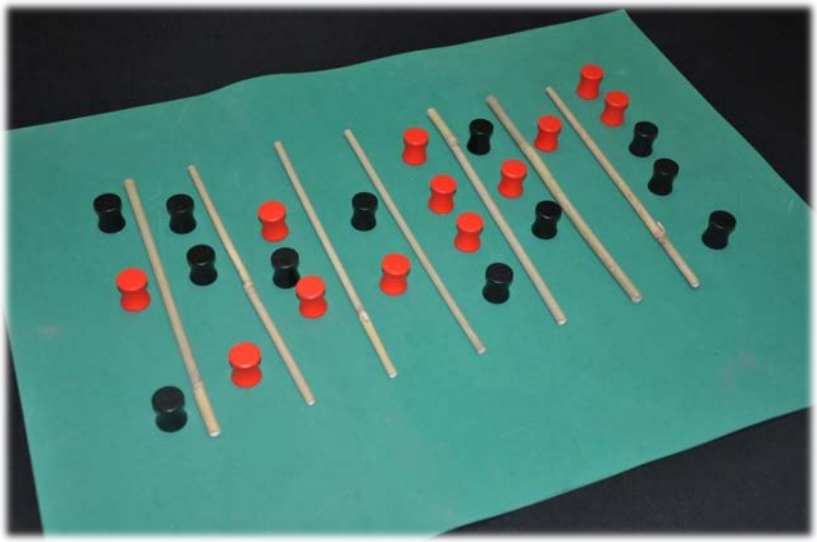
2. Linja, el juego

2.1 Visión general

Linja es un juego de mesa abstracto creado por Steffen Mülhäuser y publicado en el año 2003. Se trata de un juego para dos jugadores y se caracteriza por tener una mecánica sencilla y unas reglas fáciles de aprender, pero sin dejar de aportar altas dosis de táctica y estrategia.

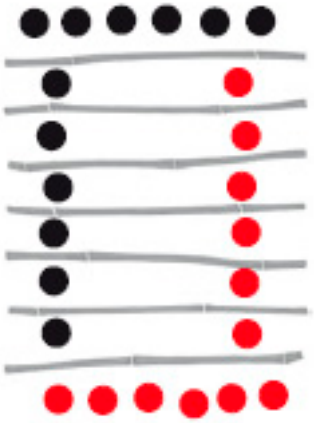
Linja se juega en un "tablero" (realmente no existe como tal, sino que sus casillas consecutivas están delimitadas por cañitas de bambú) en el cual ambos jugadores compiten por mover y hacer llegar sus fichas al extremo opuesto del tablero, intentando tenerlas más avanzadas que las del adversario en el momento en el que se desencadene el final de la partida.

Probablemente un juego con el que Linja guarde ciertas similitudes sea el Backgammon, pero a diferencia de este milenario juego en Linja el azar brilla por su ausencia. No hay intermediación de dados o cualquier otro elemento que pueda condicionar la suerte de los jugadores, sino que cada contrincante es responsable directo de la ventaja o desventaja brindadas a su adversario. Así pues, el jugador que sea capaz de jugar a más turnos vista teniendo en cuenta las posibles respuestas de su adversario será el que se alce con la victoria.



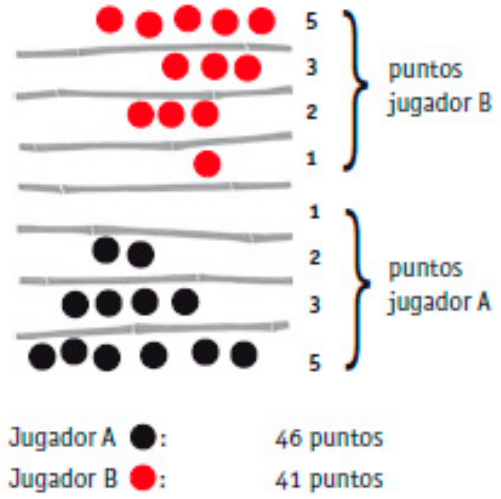
2.2 Reglas de juego

Linja se juega con veinticuatro piezas, doce de ellas para cada jugador. Siete bastoncillos de bambú delimitan las ocho casillas de juego. La disposición inicial se muestra en la figura contigua: en una de las casillas de un extremo se colocan seis fichas rojas y en la otra seis negras; las fichas restantes se colocan en las casillas intermedias, a razón de una ficha de cada color por casilla.



2.2.1 Objetivo y puntuación

El propósito de los jugadores es avanzar sus fichas hasta la casilla del extremo opuesto (la que al principio de la partida contiene las seis fichas de su adversario). La partida finaliza cuando las fichas de ambos jugadores se han sobrepasado completamente, es decir: las fichas rojas quedan a un lado del tablero y las negras a otro, sin mezclarse. En este momento se puntúa el tablero otorgando cada ficha de 5 a 0 puntos a su jugador en función de su proximidad a la casilla del extremo opuesto. El jugador con mayor puntuación es el ganador.



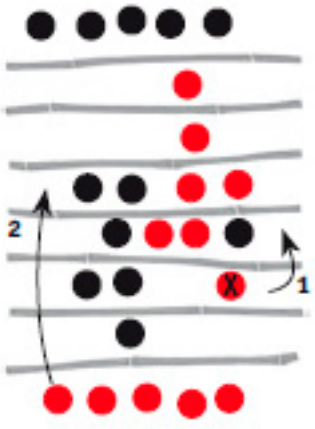
2.2.2 Movimiento

En su turno cada jugador realiza dos movimientos, conocidos como *movimiento inicial* y *segundo movimiento*.

En su *movimiento inicial* el jugador debe mover una de sus fichas exactamente una casilla hacia delante.

El número de fichas presentes en la casilla a la que ha desplazado la ficha (sin contar ésta) determinará el número de casillas que podrá mover una ficha el jugador en su *segundo movimiento*. Es decir, si en su movimiento inicial el jugador desplazó una ficha a una casilla que ya contenía tres fichas, en su segundo movimiento podrá mover una ficha tres casillas hacia delante.

Dos aclaraciones: las fichas desplazadas en ambos movimientos pueden o no ser la misma; y si en su movimiento inicial un jugador movió una ficha a una casilla vacía perderá el derecho a su segundo movimiento y deberá pasar turno (algo lógico, pudiendo entenderse como que su segundo movimiento ha sido de cero casillas).



En su movimiento inicial el jugador desplaza una ficha a una casilla que contiene cuatro fichas, por lo tanto en su segundo movimiento puede desplazar una ficha cuatro casillas.

2.2.3 Turno extra

Si en su *segundo movimiento* un jugador desplaza una ficha a una casilla vacía, dicho jugador obtendrá un turno extra completo (consistente en otro *movimiento inicial* y otro *segundo movimiento*).

Si en su turno extra el jugador desplaza de nuevo en su segundo movimiento una ficha a una casilla vacía no tendrá derecho a otro turno extra y deberá pasar turno. Es decir, no pueden concatenarse dos turnos extra seguidos del mismo jugador.

Es interesante recalcar cómo si un jugador mueve a una casilla vacía en su *movimiento inicial* pierde su segundo movimiento; pero si hace lo mismo en su *segundo movimiento* es premiado con un turno extra.

2.2.4 Excepciones a las reglas

El número máximo de fichas que puede haber en una casilla intermedia es de seis. Si una de estas casillas está completa ningún jugador puede desplazar una ficha a ellas pero sí puede pasar a través de ellas. Las dos casillas de los extremos no tienen limitación en el número de fichas que pueden contener.

Si en su movimiento inicial un jugador desplaza su ficha a la casilla del extremo opuesto, su segundo movimiento constará únicamente de un solo punto (es decir, no se tendrán en cuenta el número de fichas presentes en la casilla final para determinar los puntos de movimiento).

3. Análisis y especificación

En este capítulo se trata la parte del proyecto que resulta independiente de la tecnología. Se analizan los requisitos: qué sistema hace falta construir y qué se espera de él; y se detalla la especificación: qué tiene que hacer el sistema. También se propone un modelo del dominio que represente los elementos del "mundo real" del juego Linja.

3.1 Análisis de requisitos

Los ***requisitos funcionales*** que se prevé para la adaptación de Linja son los siguientes:

- Permitir al usuario jugar partidas contra otro jugador a nivel local (en el mismo dispositivo).
- Permitir al usuario jugar partidas contra un oponente IA, en diferentes dificultades.
- Permitir al usuario jugar partidas contra oponentes reales vía Internet.
- Permitir al usuario enfrentar dos IAs entre sí y ver cuál de las dos es mejor.
- Permitir al usuario deshacer un movimiento que haya hecho en el juego

Como ***requisitos no funcionales*** podemos citar los siguientes:

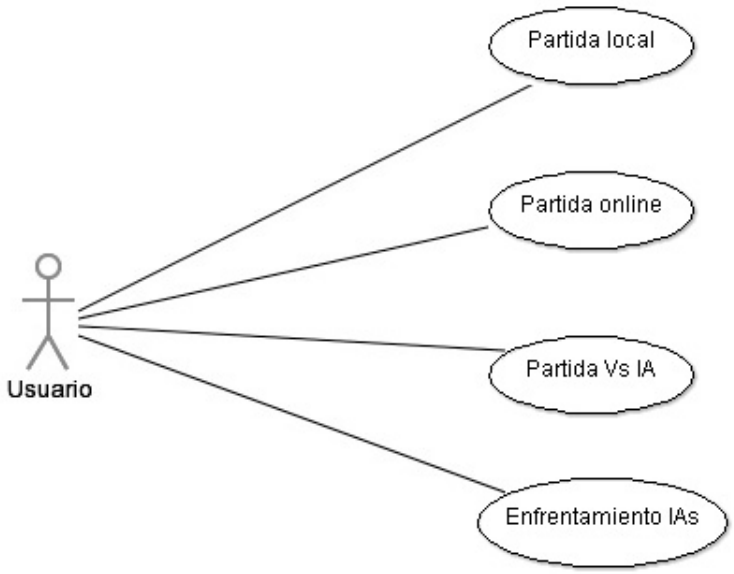
- Ofrecer una interfaz sencilla e intuitiva. Linja es un juego elegante y minimalista, el usuario debe sentir esto también en la aplicación.
- Reducir al mínimo las pantallas fuera de la partida. En un dispositivo portátil el usuario generalmente busca acceder al juego de forma rápida, sin perder excesivamente el tiempo.
- El tiempo de espera del jugador mientras el oponente de Inteligencia Artificial piensa y ejecuta su movimiento debe estar ajustado a lo que se espera en un juego de estas características en una plataforma portátil.

3.2 Modelo de casos de uso

3.2.1 Diagramas de casos de uso

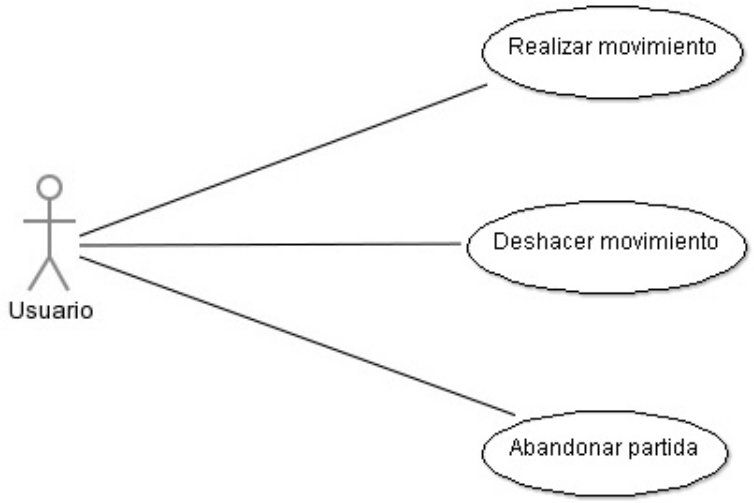
Menú Principal

Este caso correspondería al inicio típico de la aplicación: al usuario se le ofrece la selección de qué tipo de partida desea jugar (local, en línea o contra la Inteligencia Artificial) o si desea ver a dos IAs enfrentarse entre sí. El único actor del sistema es el propio usuario.



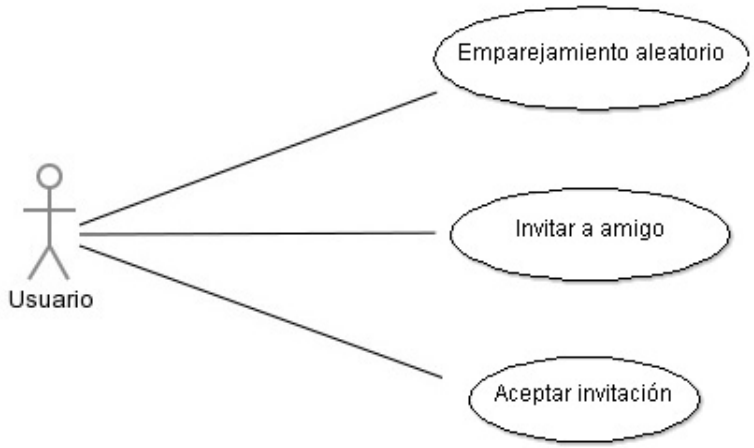
Partida

En este caso de uso se contemplan las funcionalidades que se le ofrecen al usuario en el transcurso de una partida. La más importante es realizar un movimiento en la partida (cuando sea su turno) pero el usuario también puede querer deshacer uno o varios movimientos, o abandonar la partida.



Menú online

Este caso corresponde al menú online de la aplicación, después de que el usuario haya seleccionado la opción de Partida online. Ahora debe seleccionar qué tipo de partida online desea: si emparejarse contra un oponente aleatorio que también esté buscando partida en ese momento o invitar a un amigo a una partida para jugar contra él. También, el usuario puede aceptar invitaciones que le lleguen de sus propios amigos.



3.2.2 Definición de los casos de uso

Menú Principal

Caso de uso	Partida local
Actores primarios	Usuario
Activación	El usuario selecciona la opción en el Menú Principal
Escenario principal	1. El usuario selecciona la opción de Partida local en el Menú Principal 2. El sistema muestra el tablero de juego y el jugador activo
Extensiones	-

Caso de uso	Partida Online
Actores primarios	Usuario
Activación	El usuario selecciona la opción en el Menú Principal
Escenario principal	1. El usuario selecciona la opción de Partida Online en el Menú Principal 2. El sistema ofrece al usuario escoger entre una partida contra un oponente aleatorio o partida contra un amigo. 3. El usuario selecciona el tipo de partida 4. El sistema muestra el tablero de juego y el jugador activo, cuando se han cumplido las condiciones de emparejamiento
Extensiones	2. El usuario no ha iniciado sesión: - El sistema requiere al usuario que inicie sesión - El usuario inicia sesión en el sistema

Caso de uso	Partida vs IA
Actores primarios	Usuario
Activación	El usuario selecciona la opción en el Menú Principal
Escenario principal	1. El usuario selecciona la opción de Partida vs IA en el Menú Principal 2. El sistema muestra un listado de diferentes oponentes IA y sus niveles de dificultad 3. El usuario selecciona un oponente IA y su nivel de dificultad 4. El sistema solicita el usuario quién desea que sea el jugador inicial 5. El sistema muestra el tablero de juego y el jugador activo
Extensiones	El usuario abandona la selección de IA o de jugador inicial: finaliza el caso de uso

Caso de uso	Enfrentamiento IAs
Actores primarios	Usuario
Activación	El usuario selecciona la opción
Escenario principal	1. El usuario selecciona la opción de Enfrentamiento IA en el Menú Principal 2. El sistema muestra un listado de oponentes IA y sus niveles de dificultad 3. El usuario selecciona el primer jugador IA y su nivel de dificultad 4. El sistema muestra un listado de oponentes IA y sus niveles de dificultad 5. El usuario selecciona el segundo jugador IA y su nivel de dificultad 6. El sistema muestra el tablero de juego y da comienzo la partida
Extensiones	El usuario abandona la selección de IA: finaliza el caso de uso

Partida

Caso de uso	Realizar movimiento
Actores primarios	Usuario
Activación	El usuario es seleccionado por el sistema como el jugador activo
Escenario principal	1. El sistema muestra al usuario todos los movimientos posibles 2. El usuario selecciona un movimiento 3. El sistema realiza el movimiento
Extensiones	-

Caso de uso	Deshacer movimiento
Actores primarios	Usuario
Activación	El usuario selecciona la opción
Escenario principal	1. El usuario selecciona la opción de Deshacer movimiento 2. El sistema deshace el último movimiento y vuelve a un estado anterior del tablero
Extensiones	El usuario selecciona la opción después de un turno de la IA: se deshacen todos los movimientos de la IA y el último del usuario

Caso de uso	Abandonar partida
Actores primarios	Usuario
Activación	El usuario selecciona la opción
Escenario principal	1. El usuario selecciona la opción de Abandonar la partida 2. El sistema solicita la confirmación al usuario para abandonar la partida 3. El usuario confirma el abandono de partida 4. El sistema vuelve al Menú Principal
Extensiones	El usuario cancela la petición de confirmación para abandonar la partida: finaliza el caso de uso

Menú online

Caso de uso	Emparejamiento aleatorio
Actores primarios	Usuario
Activación	El usuario selecciona la opción
Escenario principal	1. El usuario selecciona la opción de Emparejamiento aleatorio 2. El sistema agrega al jugador a una partida online 3. El sistema muestra el tablero de juego y comienza la partida
Extensiones	-

Caso de uso	Invitar a amigo
Actores primarios	Usuario
Activación	El usuario selecciona la opción
Escenario principal	1. El usuario selecciona la opción de Invitar a amigo a partida 2. El sistema muestra al usuario un listado de sus amigos 3. El usuario selecciona un amigo para invitar 4. El sistema crea la partida y muestra el tablero de juego
Extensiones	-

Caso de uso	Aceptar invitación
Actores primarios	Usuario
Activación	El usuario recibe una invitación de partida
Escenario principal	1. El sistema muestra al usuario una invitación de partida recibida 2. El usuario acepta la invitación 3. El sistema muestra el tablero de juego con la partida iniciada por el oponente online
Extensiones	El usuario rechaza la invitación a la partida. Finaliza el caso de uso.

3.3 Modelo conceptual de los datos

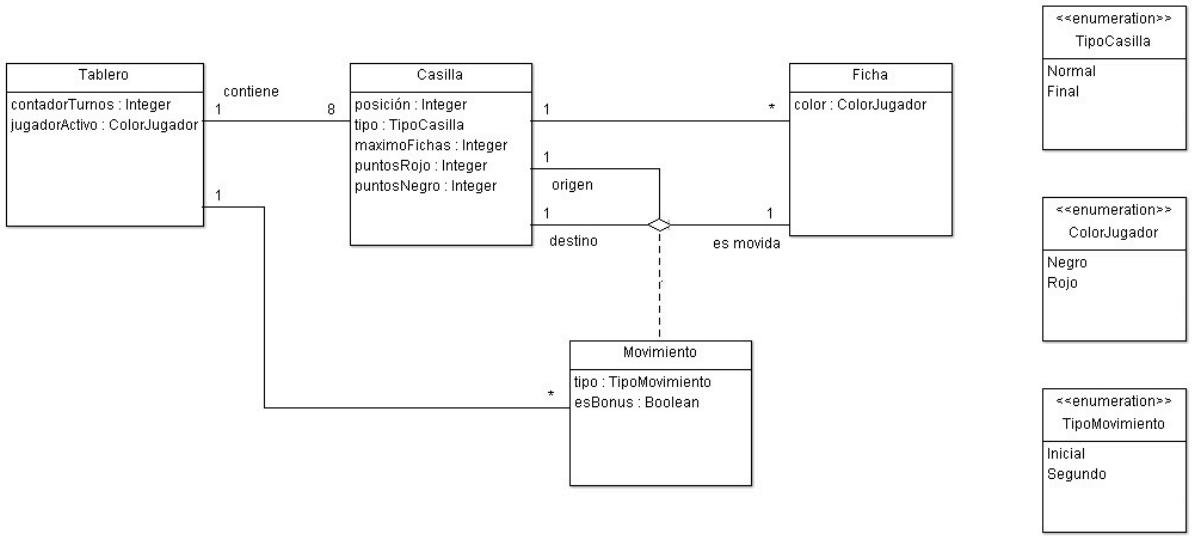
En este apartado se ofrece una propuesta de representación del modelo del dominio del juego Linja.

La clase conceptual más importante es la de tablero, pues aunque no exista un tablero físico propiamente dicho (recordemos, tan sólo unos palitos dividen las casillas) de facto lo hay y resulta más intuitivo considerarlo como tal.

El tablero contiene ocho casillas, identificada cada una por su posición en el mismo. Cada casilla es de un tipo (Final si está en uno de los extremos del tablero o Normal si se encuentra en los espacios intermedios) y puede albergar un máximo determinado de fichas (en función de su tipo -esta restricción se incluye más adelante-). También cada casilla otorga al final de la partida unos puntos al jugador rojo y/o negro según el número de fichas de cada uno presentes en la misma.

Así pues, cada casilla contiene de cero a varias fichas (se aplica la restricción anteriormente comentada) y la única diferencia entre fichas es el color, que representa a un jugador.

Una movimiento viene determinado por la ficha que se mueve y las casilla origen y destino de la misma. Cada movimiento puede tratarse del movimiento inicial del jugador o bien de su segundo movimiento; y también puede formar parte de un turno extra o no. A su vez el tablero lleva la cuenta de los movimientos en él efectuados y de quién es el jugador activo (al que le toca mover en el instante actual).



Restricciones:

a. Si la casilla es de tipo Normal el número máximo de fichas que puede albergar es 6. Si es de tipo Final no hay límite.
b. Todos los movimientos en el tablero han tenido como origen y destino casillas del mismo.

Se podrían argumentar diferentes versiones del modelo del dominio. Durante el proceso de adaptación se han tenido en cuenta numerosas alternativas que por una u otra razón se han descartado.

Por ejemplo, se podría especializar la clase Casilla y particionar discriminando por su tipo en Casilla Normal y Casilla Final. El motivo por el que se descartó fue que dichas clases no tendrían atributos adicionales distintos y que tampoco tienen comportamientos sustancialmente distintos. La restricción en el número de fichas que pueden contener no justificaría por sí sola dicha especialización.

Otros casos parecidos sería el de la especialización de Ficha por su color (Rojo o Negro) y el de Movimiento discriminando por su tipo (Inicial o Segundo movimiento) pero las razones que se adujeron son las mismas: ausencia de atributos específicos y comportamientos idénticos. Por todo ello en los tres casos se decidió simplificarlo y utilizar enumeraciones.

Por otra parte se podría sugerir la inclusión de un atributo que especificase el número de posiciones que se ha movido una ficha (en la clase Movimiento) pero ésta sería información fácilmente derivable a partir de las casillas origen y destino del movimiento.

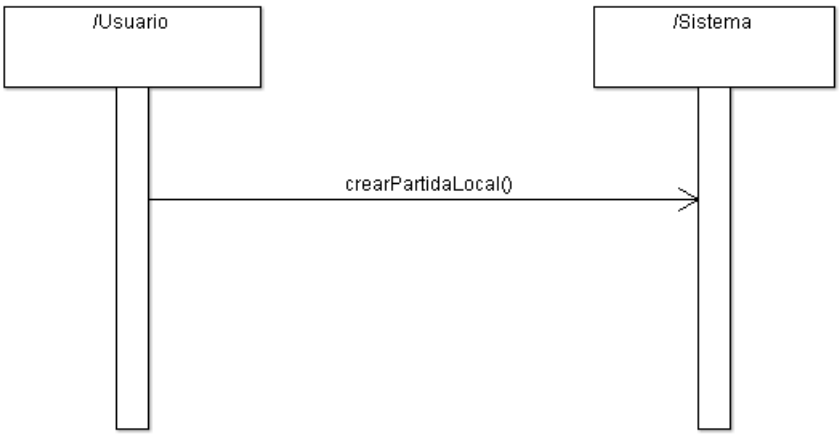
La ausencia de una clase Posición está justificada en la medida en la que el orden o disposición de las fichas dentro de una casilla es irrelevante.

3.4 Modelo de comportamiento del sistema

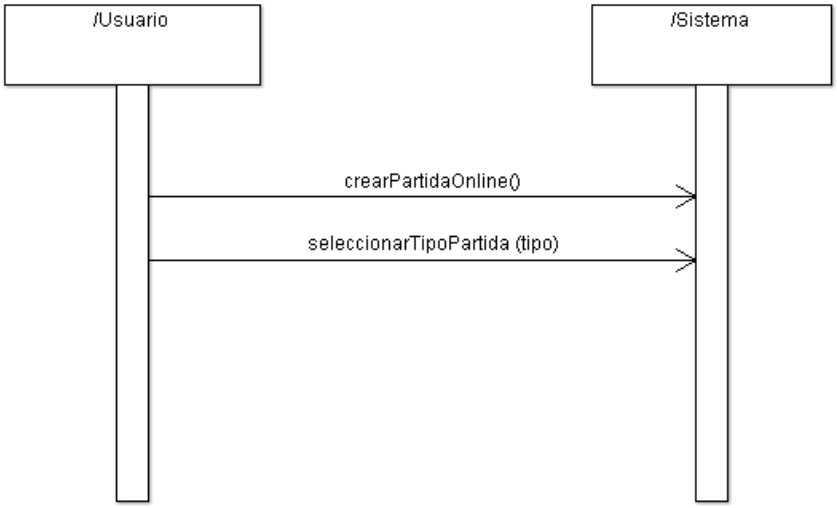
3.4.1 Diagramas de secuencia del sistema

Menú principal

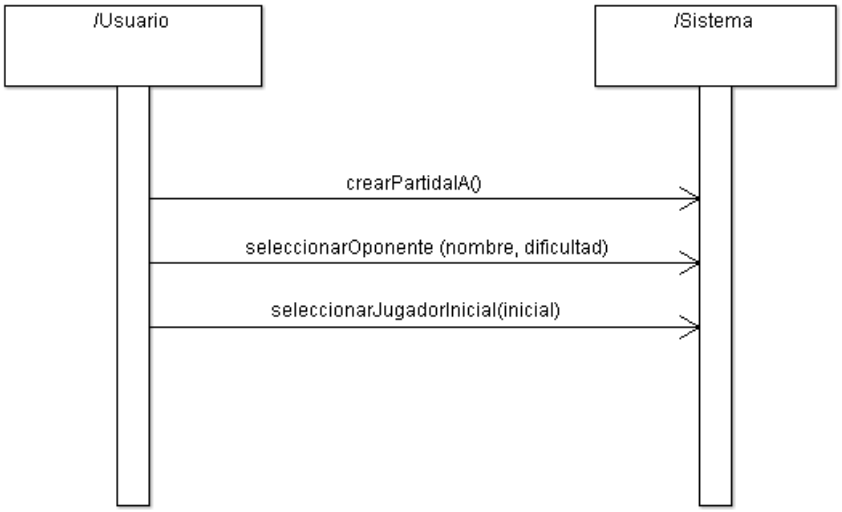
- *Partida local:* El usuario solicita al sistema crear una partida local.



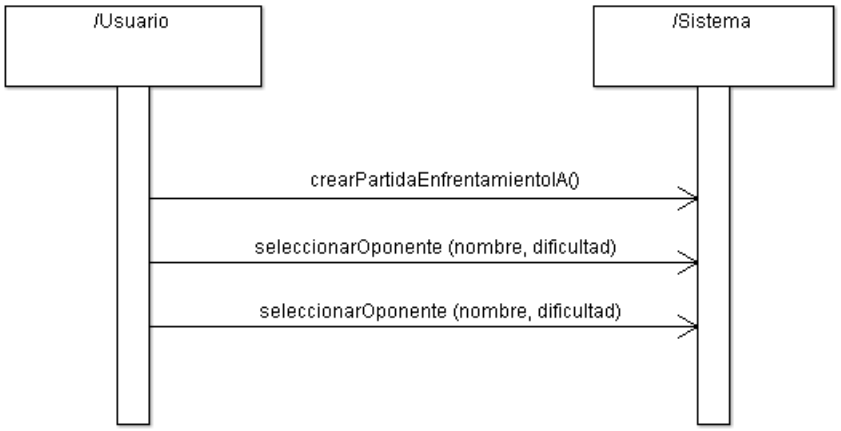
- *Partida online:* El usuario solicita al sistema crear una partida online y selecciona qué tipo de partida desea (si emparejamiento contra un rival aleatorio o contra un amigo).



- *Partida Vs IA:* El usuario solicita al sistema crear una partida contra una Inteligencia Artificial. Selecciona el oponente y su dificultad. Posteriormente selecciona si quiere ser el jugador inicial en la partida o no.

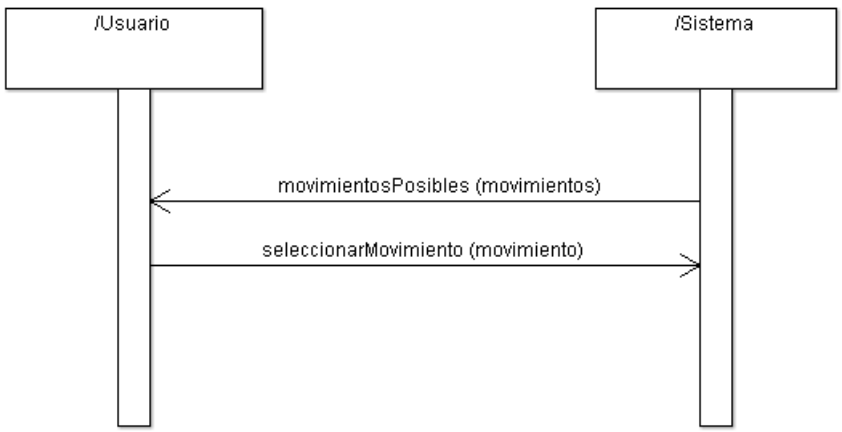


- *Enfrentamiento IAs:* El usuario solicita al sistema enfrentar a dos oponentes de Inteligencia Artificial. Primero selecciona la IA y la dificultad del primero de ellos y seguidamente hace lo mismo con el segundo.

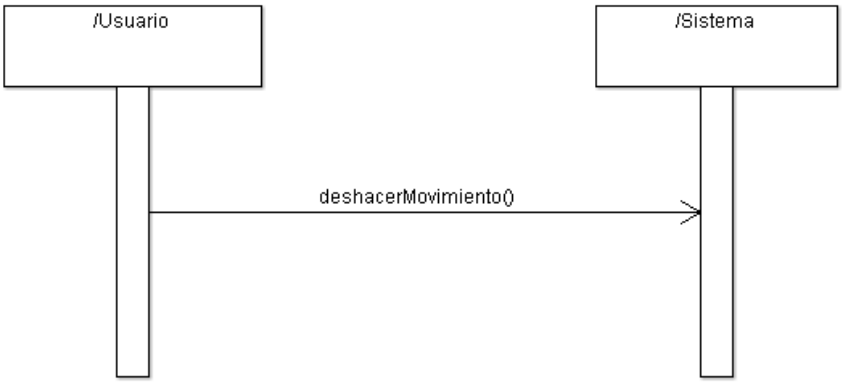


Partida

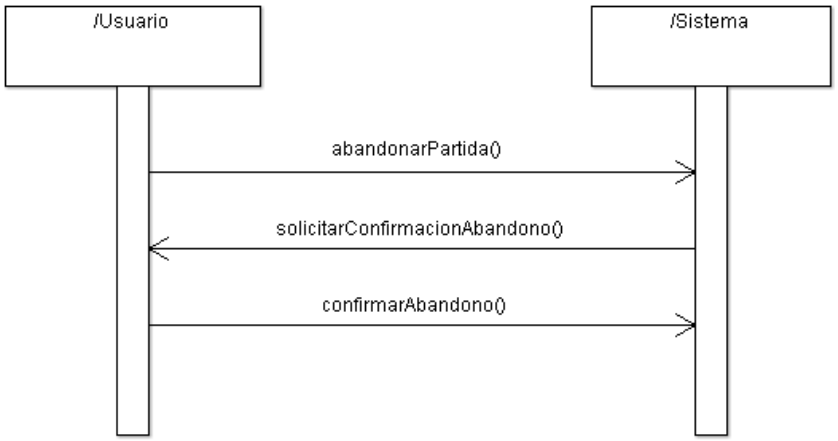
- *Realizar movimiento:* El usuario recibe del sistema una lista con los posibles movimientos que puede realizar. El usuario selecciona uno de ellos para efectuarlo. Como precondición, debe ser el turno del jugador.



- *Deshacer movimiento:* El usuario selecciona deshacer un movimiento. No tiene que especificar cuál, porque siempre se deshace el último movimiento efectuado sobre el tablero**.**



- *Abandonar partida:* El usuario indica al sistema que quiere abandonar la partida. El sistema le solicita la confirmación y el usuario debe aceptar o rechazarla**.**



4. Diseño

4.1 Arquitectura en tres capas

Para estructurar la aplicación se ha decidido optar por la típica arquitectura en tres capas:

Capa de presentación: Se encarga de la interfaz gráfica y la interacción con el usuario, capturando las peticiones del usuario y mostrándole la información generada por el sistema. Fundamentalmente trata de las ventanas de la aplicación, los menús, botones, listados, diálogos, etc.

En el caso de nuestra aplicación, esta capa es la responsable de mostrar las pantallas al usuario, recibir sus instrucciones cuando éste pulsa algún botón, mostrar el tablero de juego, permitir que el usuario pulse sobre las fichas y las desplace, actualizar la imagen del estado del tablero, etc.

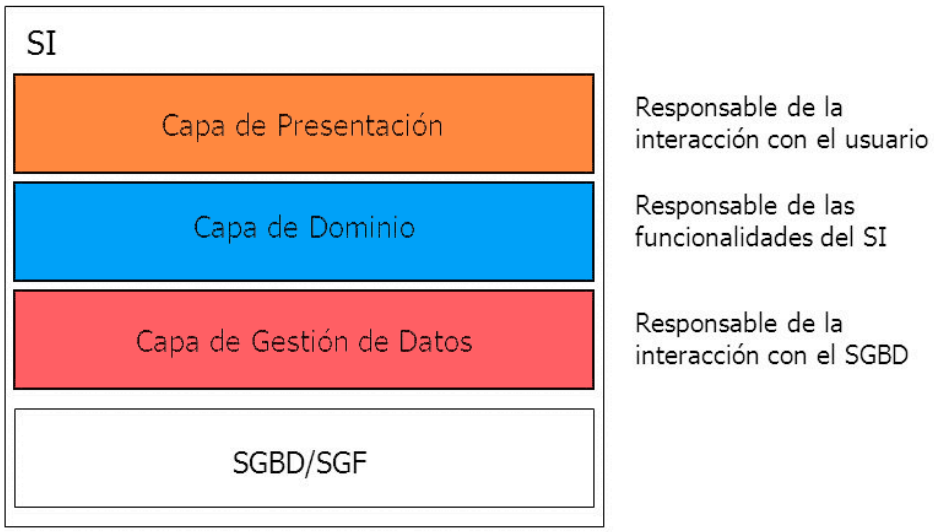
Capa de dominio: Es la responsable de la implementación de las funcionalidades del sistema: de capturar los eventos, controlar su validez, ejecutar las acciones solicitadas y cambiar el estado del dominio.

En el caso de Linja, esta capa se encarga principalmente de toda la lógica interna del juego, de calcular los movimientos posibles, de efectuar los cambios en el tablero, de obtener la información solicitada, de realizar los cálculos correspondientes a la Inteligencia Artificial, etc.

Capa de gestión de datos: Se encarga de gestionar la persistencia de los datos, de almacenarlos y tratar con la base de datos.

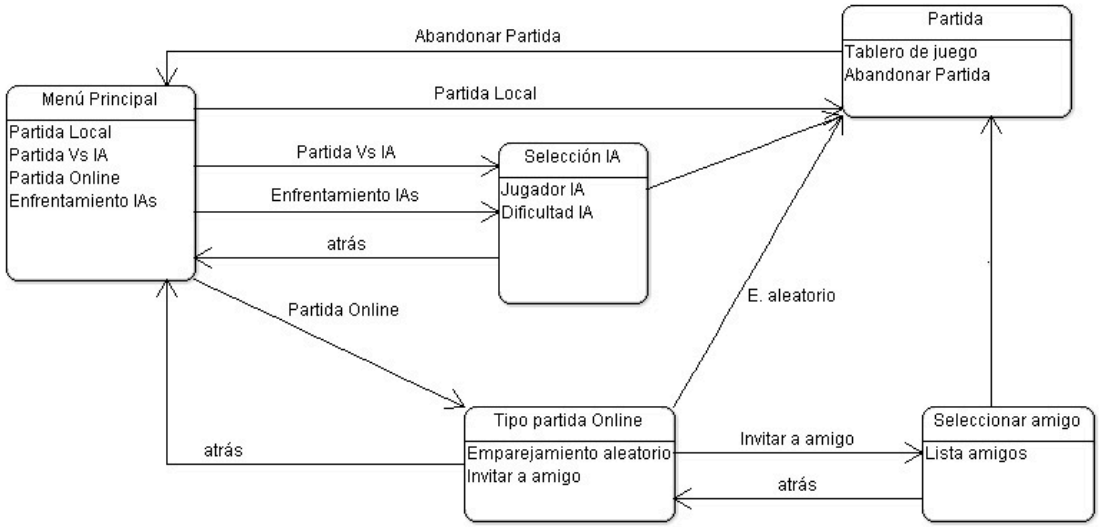
En el caso de la aplicación que nos ocupa, esta capa es prácticamente inexistente. Se podría considerar su uso si se quisieran guardar entre sesiones las preferencias del usuario o los

registros de sus partidas. En Linja sólo se usará para almacenar los amigos del usuario, de cara a poder enviarles invitaciones de partida.



4.1.1 Capa de presentación

La interfaz se ha diseñado persiguiendo que sea sencilla y amigable. El esquema de navegación entre pantallas propuesto es el siguiente:



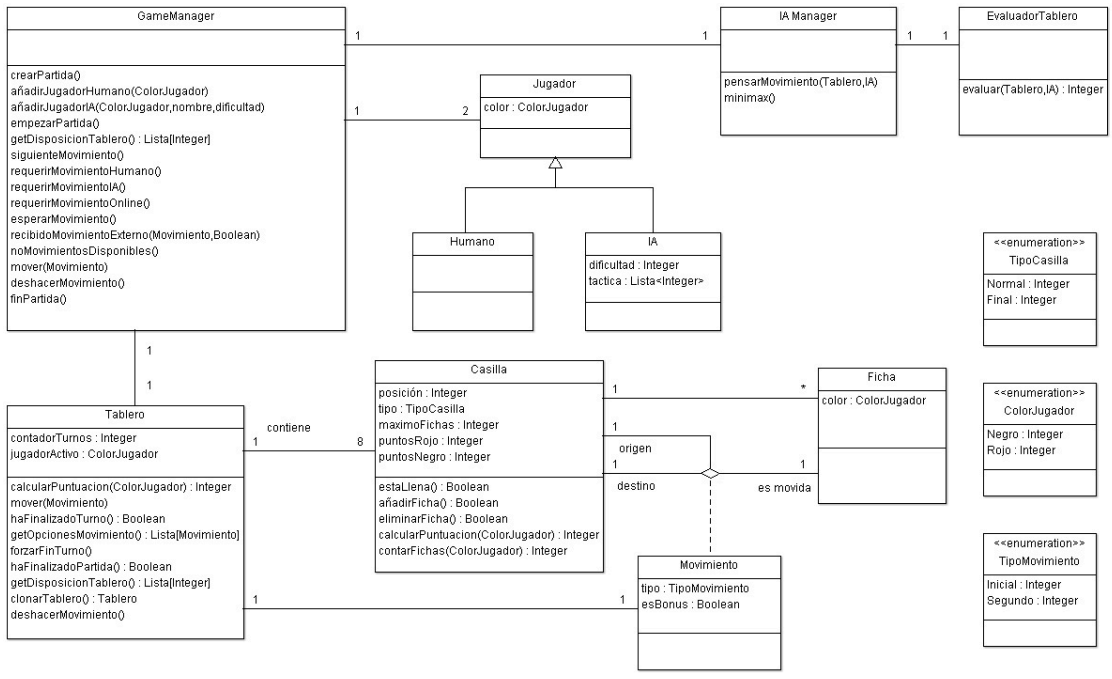
La estructura de la capa de presentación sigue una estructura Modelo-Vista-Controlador (MVC) adaptado a la solución tecnológica escogida, la cual se comentará en el capítulo de Implementación.

4.1.2 Capa de dominio

Diagrama de clases de diseño

En este apartado podemos observar el diagrama de clases de diseño, inspirado por el modelo conceptual de los datos visto en el capítulo anterior.

Para simplificar el diagrama en aras de una mayor comprensibilidad se han omitido las operaciones de consulta (*get*) y modificación (*set*) de los atributos de clase, así como los métodos privados.



La clase *GameManager* es la encargada de la gestión de la partida, desde su creación, la preparación del tablero, el control de los turnos de juego (determinando también a qué

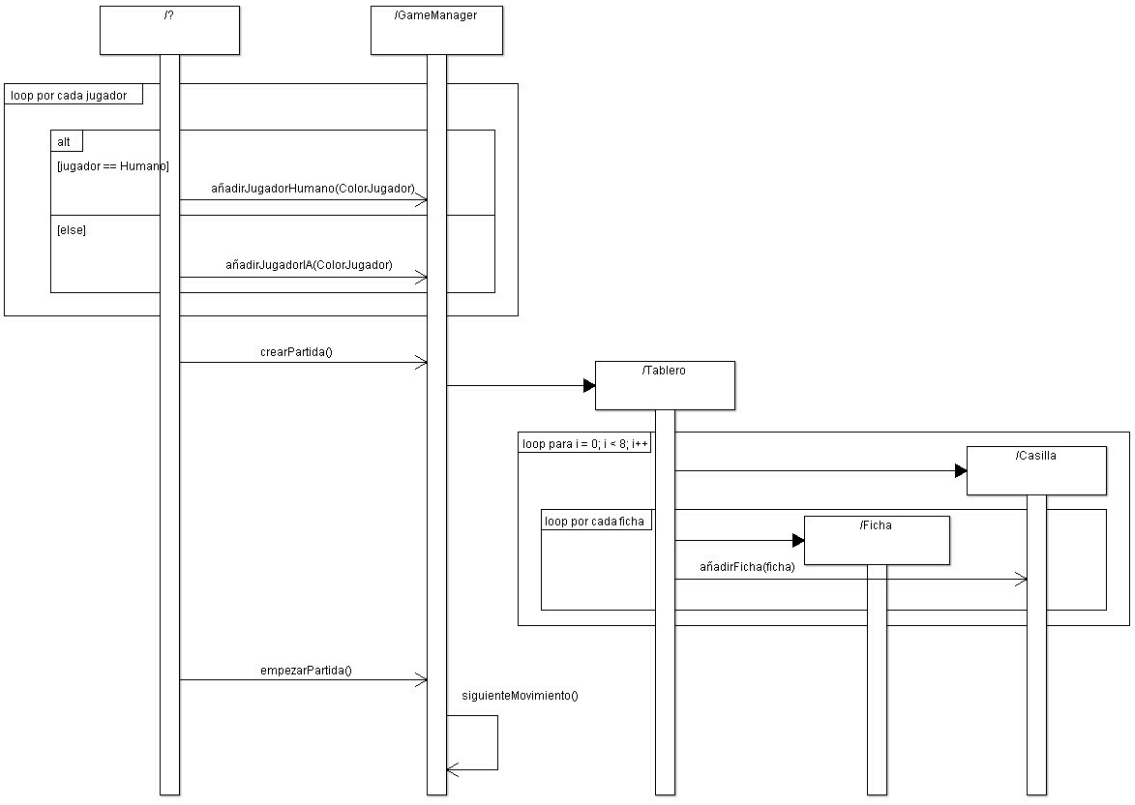
jugador le toca jugar) hasta cuándo se da la condición de fin de juego y proceder a terminar la partida.

Otras clases añadidas son *IA Manager* y *EvaluadorTablero*, sirviendo a la funcionalidad de la Inteligencia Artificial. Ambas clases se encargan de calcular el movimiento del jugador IA, lo cual se explicará en detalle en un próximo capítulo.

Finalmente también se ha añadido una clase *Jugador*, con su especialización en las clases *Humano* e *IA*; principalmente de cara a la gestión de la Inteligencia Artificial.

Diagramas de secuencia

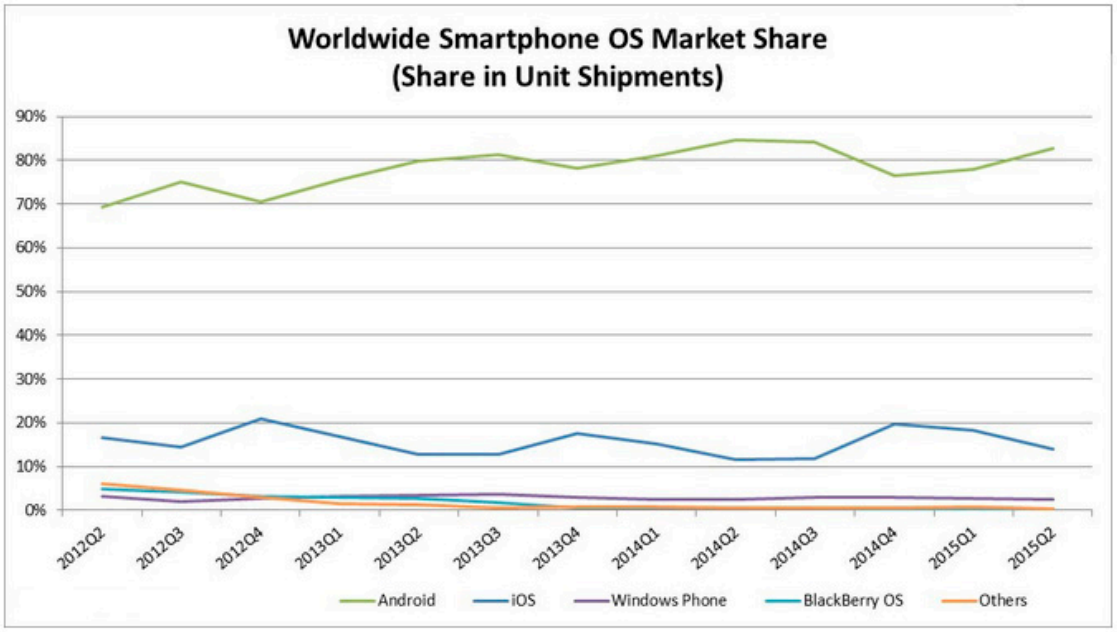
El siguiente diagrama de secuencia ilustra la funcionalidad de crear partida. Primero se añaden los jugadores y posteriormente se crea el tablero, quien a su vez se encarga de crear las casillas que lo conforman y las fichas con las que se rellenarán.



5. Implementación

Para implementar la aplicación se ha decidido utilizar Java y el SDK de Android, programando específicamente para dispositivos con este sistema operativo. Aunque existen otras plataformas que permitirían exportar el código a varios sistemas, el hecho de que Android cope actualmente cerca del 83% del mercado ha sido determinante para decidir desarrollar exclusivamente para este sistema operativo.

En el gráfico y tabla siguientes podemos ver la cuota de mercado de los diferentes sistemas a lo largo de los últimos cuatro años.



Period	Android	iOS	Windows Phone	BlackBerry OS	Others
2015Q2	82.8%	13.9%	2.6%	0.3%	0.4%
2014Q2	84.8%	11.6%	2.5%	0.5%	0.7%
2013Q2	79.8%	12.9%	3.4%	2.8%	1.2%
2012Q2	69.3%	16.6%	3.1%	4.9%	6.1%

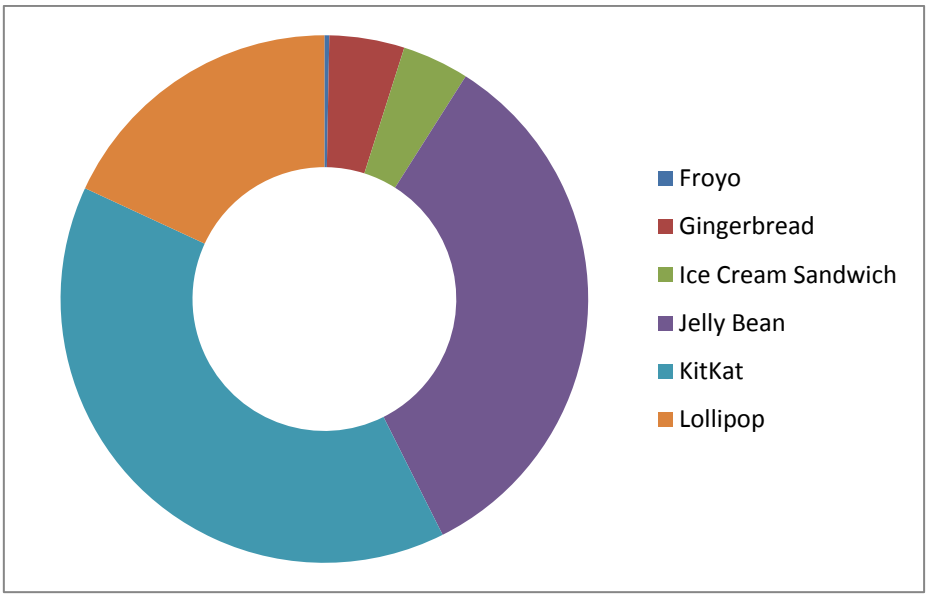
Fuente: IDC, agosto 2015 (www.idc.com)

Desde su aparición en 2008 Android ha pasado por diferentes versiones, siendo la última la 5.1 aunque está prevista la aparición de la versión 6.0 para otoño de 2015.

Esta aplicación de Linja se ha desarrollado compilando para la versión 5.0 (API 21, Lollipop), pero con soporte para versiones anteriores hasta 2.3 (API 10, Gingerbread). En la siguiente tabla y su gráfica correspondiente puede verse que cubre hasta el 99,7% de los dispositivos Android presentes actualmente entre los usuarios.

Versión	Nombre	API	Distribución
2.2	*Froyo*	8	0,3%
2.3.3-2.3.7	*Gingerbread*	10	4,6%
4.0.3-4.0.4	*Ice Cream Sandwich*	15	4,1%
4.1.x	*Jelly Bean*	16	13,0%
4.2.x		17	15,9%
4.3		18	4,7%
4.4	*KitKat*	19	39,3%
5.0	*Lollipop*	21	15,5%
5.1		22	2,6%

Datos extraídos de Google Play, principios de agosto de 2015



5.1 Clases Activity y Fragment

La clase Activity es el componente básico de cualquier aplicación Android. Se encarga de la interacción con el usuario y generalmente engloba una única acción que el usuario puede hacer. Originalmente se podía entender cada Activity como cada una de las pantallas de una aplicación.

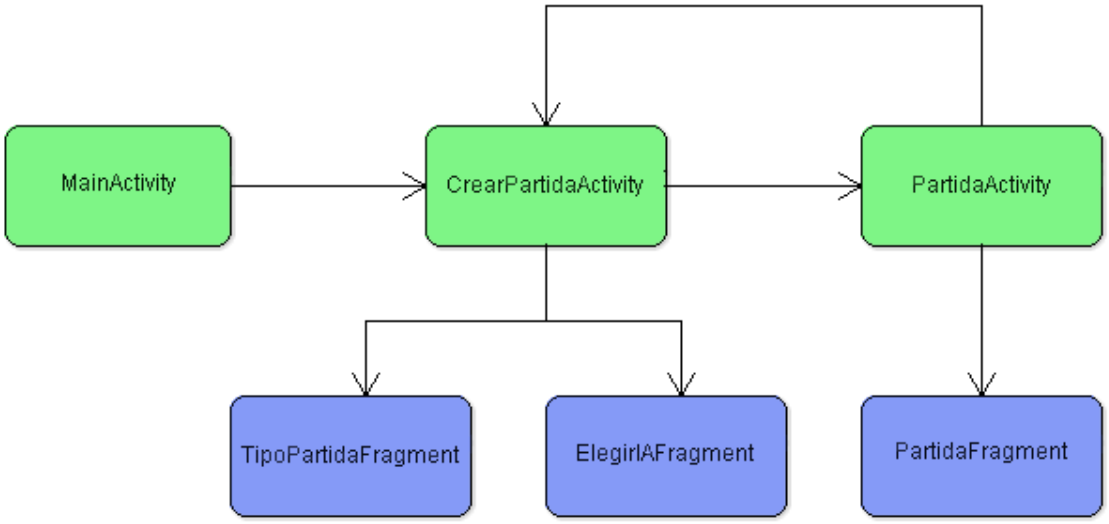
La problemática radica en que las Activities no sólo están íntimamente ligadas con la interfaz gráfica, sino que también se utilizan para el acceso a datos o la navegación a otras Activities. Esto añade cierto acoplamiento que convendrá evitar en la medida de lo posible.

Afortunadamente, a partir de la versión 3.0 de Android (API 11) se introdujo la clase Fragment. Un Fragment está siempre alojado en una Activity y representa una porción de la interfaz de usuario que puede añadirse o eliminarse de forma independiente a otros elementos que conformen dicha Activity.

Gracias a esto se consigue un bajo acoplamiento y mayor flexibilidad, añadiendo modularidad de forma que un Fragment se puede reutilizar en varias Activities sin cambiar su código. Un Fragment puede encargarse de alojar todos los elementos en pantalla tales como botones, campos de texto, imágenes, etc.; y capturar las acciones del usuario para pasárselas a su Activity correspondiente.

5.2 Estructura de la aplicación en Android

Basándonos en lo explicado en el apartado anterior y en el diseño de las pantallas de la aplicación visto en el capítulo de Diseño, se propone la siguiente estructura de Activities y Fragments:



Al iniciar la aplicación, ésta carga la *MainActivity*, quien directamente da paso a la Activity de *CrearPartida*. Es en esta donde se gestionan todas las pantallas que ve el usuario previamente a una partida: la de escoger qué tipo de partida quiere y la de elegir el oponente IA y su dificultad, cada una representada por su Fragment correspondiente. Cuando todo está listo para la partida se carga la *PartidaActivity*, la cual aloja el *PartidaFragment*, encargado de mostrar el tablero de juego y capturar los movimientos realizados por el usuario.

La Activity de Partida se comunica directamente con la capa de dominio a través de la clase *GameManager* vista en el capítulo anterior.

Las clases relacionadas con el juego online se han omitido puesto que se tratarán con más profundidad en un capítulo posterior.

5.3 Librerías externas utilizadas

Además del SDK de Android, se han utilizado otras librerías externas para añadir ciertas funcionalidades:

Android Support Library: Librería que proporciona compatibilidad de funciones actuales de Android para versiones anteriores del sistema operativo. En este proyecto se integra para permitir utilizar Fragments (introducidos en Android 3.0), manteniendo la compatibilidad con versiones antiguas.

Google Play Services: Librería que permite integrar servicios de Google tales como Google Maps, Google+, etc. En este proyecto se utiliza para obtener las funcionalidades deseadas del modo online de juego. Esto será explicado más adelante.

Gson: Librerías que permiten la serialización de objetos de objetos Java y su representación en notación Json. Se utilizan en el modo online del juego para codificar los objetos y enviar los datos a través de la red.

5.4 Implementación del multijugador

5.4.1 Google Play Game Services

Para añadir el modo multijugador a la aplicación se ha decidido utilizar la API de Google Play Game Services, un servicio de Google que ofrece las funcionalidades típicas de los juegos online de hoy en día, facilitando y simplificando su desarrollo.

Este servicio ofrece soporte tanto para juegos en tiempo real como por turnos. En el caso de Linja sólo será necesario utilizar esto último. También aporta la posibilidad de añadir rankings, logros, eventos, etc., pero en esta versión de Linja se ha decidido por no implementarlos.

Google Play Game Services libera al desarrollador de crear su propio servidor de juego para gestionar los emparejamientos de los jugadores e implementar una lista de amigos; cuando un usuario busca una partida la aplicación accede a los servidores de Google solicitando que le asigne un jugador que cumpla los criterios establecidos.

5.4.2 Consideraciones previas

Para poder desarrollar con Google Play Services se requiere el pago de una tasa de 25$ a Google. Dicha tasa permite también la publicación de aplicaciones en la Google Play Store y a diferencia de otros sistemas (como Apple Store) se trata de un pago único y no periódico.

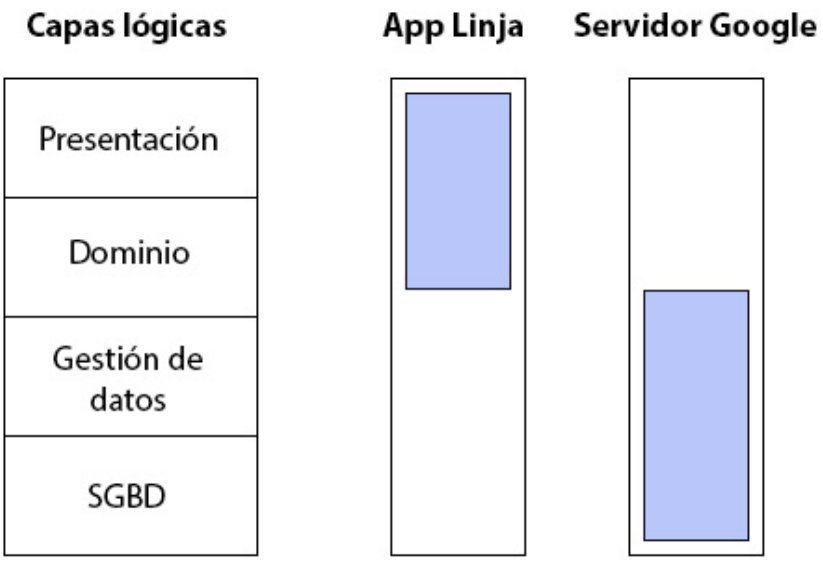
Uno de los posibles inconvenientes que presenta Google Play Game Services es que para que el usuario pueda hacer uso es necesario que inicie sesión en Google+. Actualmente todos los usuarios Android tienen una cuenta de Gmail asociada al dispositivo pero no todos ellos tienen por qué tener una cuenta de Google+. Aunque crear una cuenta se puede hacer

inmediatamente en el momento que la aplicación lo requiera para acceder al multijugador, algunos usuarios pueden ser reacios a registrarse en dicha red social.

Otro de los aspectos que hay que considerar es el límite de cuota que Google Play Services impone para cada uno de los servicios utilizados. En el caso de Game Services el límite de cuota es de 50.000.000 solicitudes por día y en el servicio de inicio de sesión en Google+ es de 20.000.000 solicitudes/día. Estas son unas cifras que hoy por hoy se antojan imposibles de alcanzar en la aplicación de Linja, puesto que exigirían un número extremadamente alto de usuarios al día usando la aplicación. Superadas estas cantidades de peticiones habría que solicitar formalmente un aumento de la cuota, quedando a consideración de Google el permitirlo; quien sólo lo haría si la aplicación ofreciese una genuina experiencia al usuario y cumpliese con su código de buenas prácticas.

5.4.3 Estructura en capas del multijugador

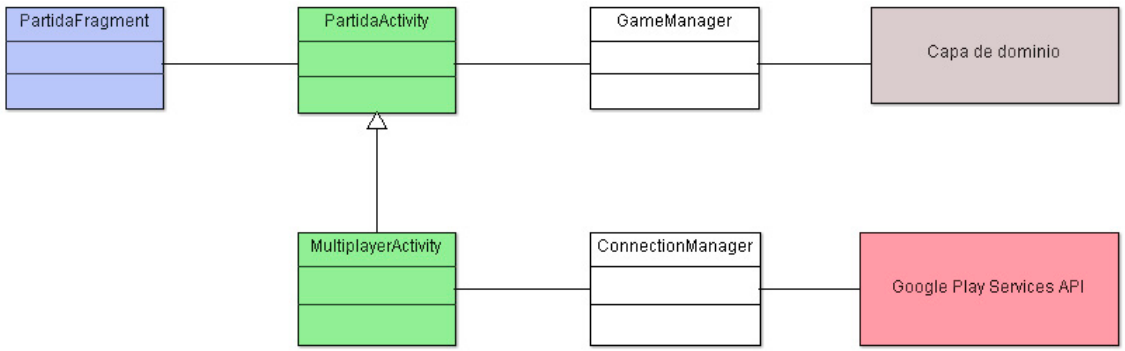
Optando por la solución ofrecida por Google Play, la distribución de las capas lógicas de la arquitectura en tres capas quedaría distribuida de la siguiente forma:



La propia aplicación instalada en el dispositivo se seguiría encargando de los elementos de las capas de presentación y gran parte del dominio pero delega exclusivamente la responsabilidad de la capa de datos a los servidores de Google Play Services (por ejemplo, la gestión de los amigos del usuario) y una parte del dominio (funcionalidad de realizar emparejamientos, etc.).

5.4.4 Estructura del modelo

La integración del modo online se ha realizado siguiendo este esquema.



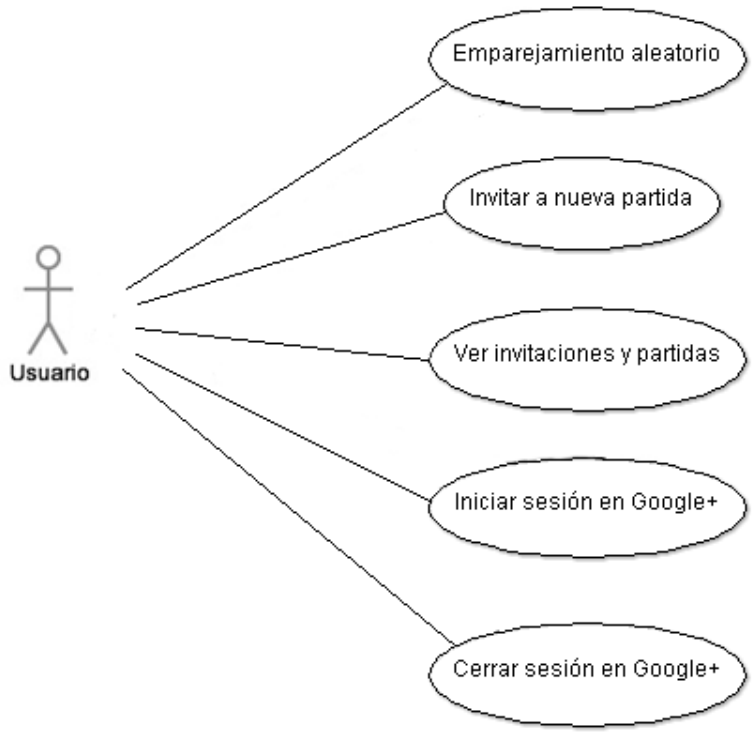
Se ha añadido una nueva Activity (*MultiplayerActivity*) que hereda de la anteriormente vista *PartidaActivity*. Utiliza los métodos propios de una partida local pero sobrescribiendo algunos para cambiar el comportamiento en partidas multijugador. Además añade métodos nuevos necesarios para gestionar el estado de la conexión y eventualmente el inicio de sesión del usuario si es requerido.

Cuando el jugador local realiza un movimiento, la clase *ConnectionManager* serializa el objeto que representa el tablero resultante y utiliza la API para enviarlo. El otro jugador recibe el objeto, lo deserializa y sustituye en su capa de dominio su tablero actual por el recibido; además de mostrar los cambios por pantalla.

5.4.5 Modificación de los casos de uso

Al implementar el modo multijugador usando las librerías de Google Play Game Services se ha considerado necesario cambiar los casos de uso de la especificación, especialmente los del modo online.

La integración del inicio de sesión en Google+ exige introducir dos funcionalidades adicionales: la del propio inicio de sesión (si el usuario no está conectado) y la desconexión (si se encuentra conectado). También se ha modificado el caso de uso que muestra las partidas activas del usuario y sus invitaciones; y el caso de uso de invitar a un oponente a una partida.



Los casos de uso modificados son los siguientes:

Caso de uso	Invitar a nueva partida
Actores primarios	Usuario
Prerrequisito	El usuario ha iniciado sesión en Google+
Activación	El usuario selecciona la opción
Escenario principal	1. El usuario selecciona la opción de Invitar a nueva partida 2. El sistema muestra al usuario un listado de oponentes (amigos suyos, oponentes recientes, jugadores cercanos). 3. El usuario selecciona un jugador para invitar 4. El sistema crea la partida y muestra el tablero de juego
Extensiones	-

Caso de uso	Ver invitaciones y partidas
Actores primarios	Usuario
Prerrequisito	El usuario ha iniciado sesión en Google+
Activación	El usuario selecciona la opción
Escenario principal	1. El usuario selecciona la opción de Ver invitaciones y partidas 2. El sistema muestra al usuario un listado de partidas iniciadas e invitaciones a partidas. 3. El usuario selecciona una partida o invitación para cargar el juego
Extensiones	-

Caso de uso	Iniciar sesión en Google+
Actores primarios	Usuario
Prerrequisito	El usuario **no** ha iniciado sesión en Google+
Activación	El usuario selecciona la opción
Escenario principal	1. El usuario selecciona la opción de iniciar sesión 2. El sistema solicita al usuario que introduzca su email y contraseña 3. El usuario accede e inicia sesión
Extensiones	2. Si el sistema tiene almacenados los datos de inicio de sesión, no será necesario que se los solicite al usuario

Caso de uso	Cerrar sesión en Google+
Actores primarios	Usuario
Prerrequisito	El usuario ha iniciado sesión en Google+
Activación	El usuario selecciona la opción
Escenario principal	1. El usuario selecciona la opción de cerrar sesión 2. El sistema cierra la sesión activa del usuario
Extensiones	-

Además, el caso de uso de Partida online del Menú Principal también se ve alterado. La nueva versión es la siguiente:

Caso de uso	Modo online
Actores primarios	Usuario
Activación	El usuario selecciona la opción en el Menú Principal
Escenario principal	1. El usuario selecciona la opción de Modo online en el Menú Principal 2. El sistema muestra al usuario el menú del Modo online
Extensiones	-

6. Implementación de la Inteligencia Artificial

En este capítulo se abordará el desarrollo de una inteligencia artificial para el juego Linja. Principalmente se analizará qué tipo de juego es Linja desde el prisma de la teoría de juegos, se escogerá un conocido algoritmo de inteligencia artificial y se propondrá una adaptación eficiente del mismo según las particularidades que presenta Linja.

6.1 Análisis previo

6.1.1 Linja y la teoría de juegos

Aunque el término de teoría de juegos se suele asociar al campo de la economía, la sociología o la estrategia militar; se trata de una disciplina matemática que estudia los juegos abstractos e idealizados desde el punto de vista del comportamiento previsto y observado de sus individuos, y de las estrategias óptimas que pueden seguir. A continuación estudiaremos la tipología de Linja según los objetivos de los jugadores y la información que cada uno de ellos tiene sobre el juego.

Linja como juego de suma cero

Se pueden definir como juegos de suma cero aquellos en los que las ganancias y las pérdidas de todos los jugadores implicados están balanceadas entre sí. Es decir, lo que gane un jugador restado a lo que pierdan los demás jugadores da como resultado cero.

Por el contrario, en los juegos de suma no nula este equilibrio no está presente, las ganancias y pérdidas de los jugadores no están compensadas y un jugador puede ganar o perder independientemente de que pierda o gane el otro jugador.

Teniendo en consideración que en un juego de mesa no cooperativo la victoria puede entenderse como la única ganancia posible y la derrota como la única perdida posible, podríamos considerar a Linja como un juego de suma cero. En Linja, como en otros juegos como el ajedrez o el Go, que un jugador gane implica que el otro jugador ha perdido, evidentemente, y viceversa. Por lo tanto, adoptar una estrategia que conduzca a la propia victoria puede ser tan válido como seguir otra estrategia que se centre exclusivamente en provocar la derrota del oponente.

Conviene puntualizar que este último planteamiento no solamente se debe a que Linja sea un juego de suma cero sino a que también se trata de un juego de dos jugadores. En un juego de suma cero con tres o más jugadores, centrarse en causar la derrota de uno de los adversarios o de todos probablemente sea más complejo que intentar seguir una estrategia que nos lleve a la victoria.

Linja como juego de información perfecta

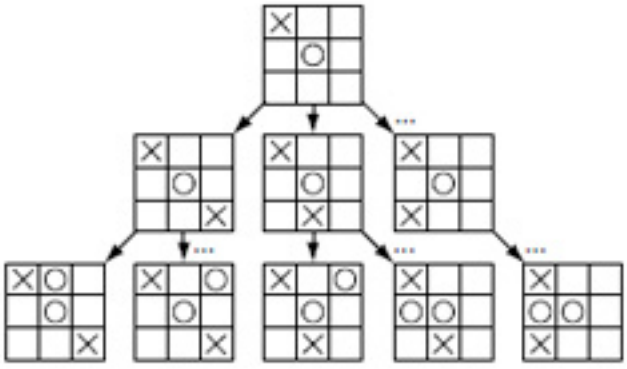
Se conocen como juegos de información perfecta aquellos en los que los jugadores saben en cada momento todo sobre el estado de la partida. Es decir, cuando un jugador debe tomar una decisión en forma de acción, sabe qué abanico de opciones de movimiento ofrecerá a su oponente, y a su vez qué nuevas opciones le reportará la decisión tomada por su oponente.

Como se habrá podido fácilmente deducir Linja se trata de un juego de información perfecta. En todo momento sabemos qué disposición tendrá el tablero después de uno de nuestros movimientos: cuántos puntos de movimiento le vamos a ofrecer al adversario, qué acciones nos brindarán un turno extra, cuáles harán que perdamos el segundo movimiento, etc.

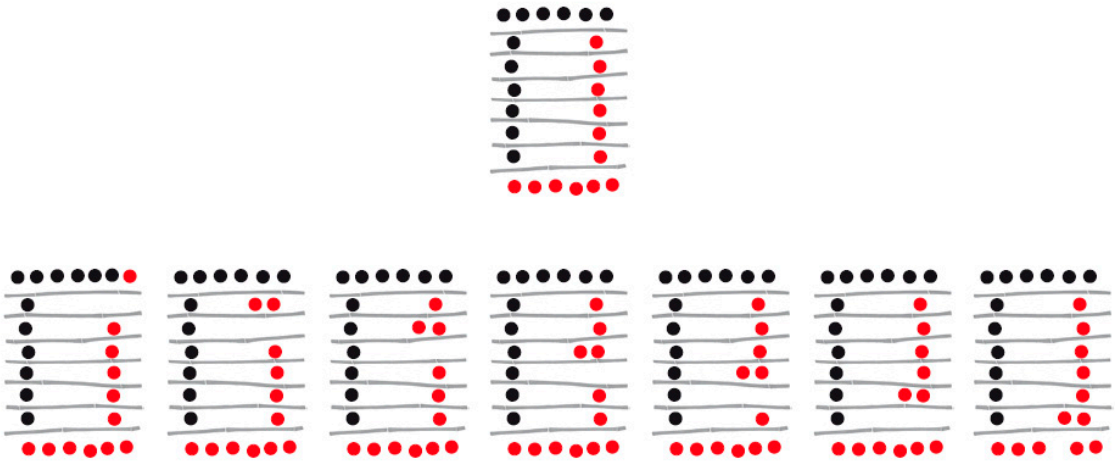
6.1.2 Representación del árbol de juego

Una forma óptima de representar la toma de decisiones en un juego de mesa es a través de un árbol de juego, donde cada nodo representa un estado posible del tablero y el número de ramas todas las posibles decisiones que puede tomar el jugador en dicho momento.

La figura siguiente muestra un ejemplo de árbol de juego de Tic Tac Toe. Cada nivel vertical representa el turno de un jugador, mientras que cada horizontal contiene las opciones de movimiento posibles que se le ofrecen (aunque los puntos suspensivos obvian otros movimientos).



En la figura mostrada a continuación podemos observar un árbol de juego de Linja que contiene todas las alternativas de movimiento que se le ofrecen al jugador rojo, en caso de ser éste el jugador inicial, al comienzo de la partida. Corresponde al movimiento inicial de desplazar una ficha una casilla.



Teniendo en cuenta que dada una casilla con dos o más fichas el movimiento de una de ellas conducirá siempre, a efectos de las reglas de juego, al mismo resultado que mover cualquier otra de esa misma casilla, se puede afirmar que cualquier estado del tablero de Linja ofrece como máximo **siete** decisiones distintas. Este sería pues el factor de ramificación, o *branching factor*, máximo de cada nodo. Aunque este factor puede verse alterado si alguna casilla interior del tablero está completa y bloquea la posibilidad de desplazar una ficha a ella o si hay alguna casilla en la que el jugador no tenga ninguna ficha.

6.2 Algoritmo de IA

Para responder a la necesidad de crear una Inteligencia Artificial para nuestro juego Linja se ha decidido utilizar el algoritmo Minimax, uno de los más famosos para juegos por turnos y utilizado durante años en el desarrollo de IAs para juegos como el ajedrez, Go, etc.

6.2.1 Minimax

El funcionamiento del algoritmo Minimax es sencillo: dada una disposición del tablero explora todas las posibles combinaciones de movimiento hasta cierto número determinado de turnos vista (lo que se denomina la *profundidad* del algoritmo) y escoge la que considera mejor, basándose en la premisa de que en su turno nuestro oponente elegirá el movimiento que más nos perjudique (en aras de *minimizar* nuestra puntuación) y en nuestro turno nosotros siempre realizaremos el movimiento que nos resulte más beneficioso (intentando *maximizar* la puntuación).

Aunque el algoritmo Minimax es una opción óptima para implementar una IA para un juego de este tipo, la clave del éxito radica en saber valorar adecuadamente la puntuación de cada estado del tablero. De nada serviría tener un algoritmo optimizado si luego erramos a la hora de indicarle la lógica de qué movimientos son mejores que otros. Este es el trabajo de la función heurística de evaluación, de la que hablaremos más adelante en otro capítulo.

Ahora veremos una implementación básica del algoritmo Minimax y cómo podemos adaptarla para que resulte más eficiente en Linja.

Implementación básica

El código en la página siguiente muestra el algoritmo Minimax propuesto por Ian Millington en su libro *Artificial Intelligence for Games*.

Se nos presenta una función recursiva que recibe el estado actual del tablero, el jugador al que le toca mover y la profundidad del algoritmo (esto es, como hemos dicho antes, a cuántos movimientos vista queremos jugar).

Si el estado actual del tablero ya representa una partida finalizada o si ha alcanzado la profundidad máxima, evaluamos y obtenemos la puntuación actual según la función heurística y devolvemos el resultado. De no ser así, realizamos diversas llamadas recursivas a la función Minimax por cada movimiento posible en el estado actual; al obtener la valoración nos quedamos con la que más nos convenga según la premisa *minimax*: si es el turno de nuestro oponente escogemos el movimiento que minimice nuestra puntuación y si es nuestro turno el que la maximice.

```
Minimax (tablero, jugador, maxProfundidad, actualProfundidad){
      Si (tablero.gameOver( ) || actualProfundidad == maxProfundidad){
            Retornar {tablero.evaluar(jugador),  nada }
      } si no {
            mejorMovimiento = nada;
            Si (tablero.jugadorActual ( ) == jugador){
                  mejorPuntuacion = -INFINITO;
            } si no {
                  mejorPuntuacion = INFINITO;
            }
            Para (movimiento en tablero.obtenerMovimientos( )){
                  nuevoTablero = tablero.mover(movimiento);
                  {actualPuntuacion, actualMovimiento} =
                        Minimax(nuevoTablero, jugador, maxProfundidad,
                        actualProfundidad + 1);
```

```
            Si(tablero.jugadorActual( ) == jugador){
                  Si(actualPuntuacion > mejorPuntuacion){
                        mejorPuntuacion = actualPuntuacion;
                        mejorMovimiento = movimiento;
                  }
            } si no {
                  Si (actualPuntuacion < mejorPuntuacion){
                        mejorPuntuacion = actualPuntuacion;
                        mejorMovimiento = movimiento
            }}}
      Retornar {mejorPuntuacion, mejorMovimiento}
```

Implementación adaptada

La implementación propuesta en la página anterior es perfectamente válida para utilizarla en Linja, pero ciertas características propias del juego la convierten en altamente ineficiente.

La versión del algoritmo Minimax presentada por Ian Millington está pensada para juegos en los que los jugadores realizan turnos de un solo movimiento, como el ajedrez o el Go. En estos casos a la función Minimax le basta con devolver únicamente el mejor movimiento inmediato a realizar.

En el caso de Linja los jugadores generalmente realizan en su turno dos movimientos: el movimiento inicial y el segundo movimiento. Incluso a veces llegan a obtener un turno extra, por lo que acaban moviendo cuatro veces seguidas. Con la implementación de Millington haríamos una llamada al algoritmo Minimax por cada movimiento a calcular: dos veces en un turno normal o cuatro si se ha obtenido un turno extra.

Como se ha dicho, esto sería ineficiente porque ante la ausencia de movimientos del oponente o del factor azar, el resultado devuelto como mejor opción para un movimiento inicial sería el mismo que el resultado devuelto para su segundo movimiento.

La solución radicaría en ejecutar el algoritmo una sola vez para obtener los dos o –en caso de turno extra- cuatro movimientos siguientes. Para conseguir este resultado habría que realizar cambios en el algoritmo de forma que no sólo se devolviera el primer próximo movimiento, sino todos los movimientos que conducen al nodo hoja con mayor evaluación. Con este vector de movimientos la IA obtendría todos los que debe efectuar en este turno, evitando que realice diversas llamadas al algoritmo Minimax que no harían más que incrementar el coste de ejecución.

El inconveniente que presenta esta implementación es que por cada movimiento extra obtenido, el cálculo del mismo se ha efectuado reduciendo la profundidad del algoritmo Minimax en uno respecto a su antecesor. Es decir, para el cálculo de un movimiento inicial con profundidad 5, la profundidad del segundo movimiento devuelto será de 4. Si además el usuario obtuviera un turno extra, la profundidad del nuevo movimiento inicial sería de 3 y la del segundo movimiento de 2.

No obstante, ante las ventajas e inconvenientes de dicha implementación en comparación con la implementación presentada por Millington, se ha considerado conveniente realizar los ajustes presentados en este apartado. Así pues, la solución propuesta es la que sigue.

```
Minimax (tablero, jugador, maxProfundidad, actualProfundidad){
      Si (tablero.gameOver( ) || actualProfundidad == maxProfundidad){
            PuntuacionMovimientos vector[maxProfundidad]
            vector[maxProfundidad] = {tablero.evaluar(jugador), nada}
            Retornar vector;
      } si no {
            mejorMovimiento = nada;
            Si (tablero.jugadorActual ( ) == jugador){
                  mejorPuntuacion = -INFINITO;
            } si no {
                  mejorPuntuacion = INFINITO;
            }
            PuntuacionMovimientos vector[maxProfundidad];
            Para (movimiento en tablero.obtenerMovimientos( )){
                  nuevoTablero = tablero.mover(movimiento);
                  {actualPuntuacion, actualMovimiento} =
                        Minimax(nuevoTablero, jugador, maxProfundidad,
                        actualProfundidad + 1);
                  vector[actualProfundidad] = {actualPuntuacion,
                        actualMovimiento}
                  Si(tablero.jugadorActual( ) == jugador){
                        Si(actualPuntuacion > mejorPuntuacion){
                              mejorPuntuacion = actualPuntuacion;
                              mejorMovimiento = movimiento;
                              vector = {actualPuntuacion,
                        actualMovimiento}
                        }
                  } si no {
                        Si (actualPuntuacion < mejorPuntuacion){
                              mejorPuntuacion = actualPuntuacion;
                              mejorMovimiento = movimiento;
                              vector = {actualPuntuacion,
                        actualMovimiento}

                  }}}
```

```
vector[actualProfundidad] = {mejorPuntuacion, mejorMovimiento}
Retornar vector;
```

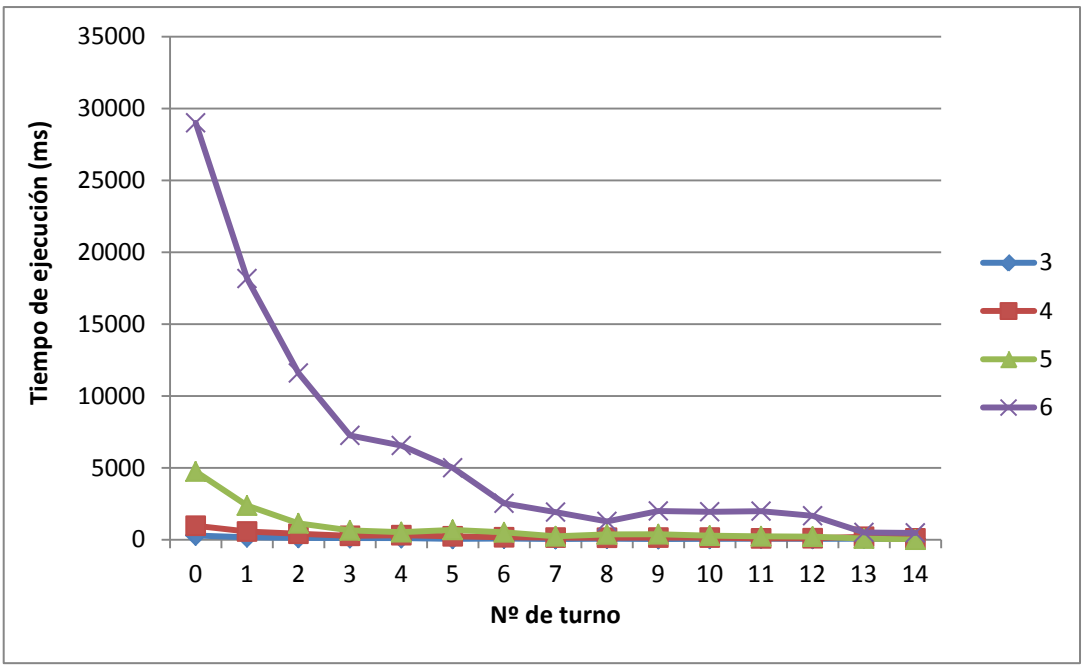
Evaluación de la implementación adaptada

El siguiente gráfico muestra el coste de tiempo de ejecución del Minimax para diversas profundidades del algoritmo, en su implementación adaptada. Como se puede observar el coste decrece a medida que avanza el juego; esto es así por dos motivos:

El primero, porque cuanto más cerca se está del final de la partida algunos nodos del árbol de juego pueden presentar ya un estado de partida acabada, sin necesidad de haber alcanzado la profundidad máxima del algoritmo.

El segundo, porque el estado del tablero justo antes de comenzar la partida da lugar a que la primera ejecución del algoritmo Minimax tenga un factor de ramificación de siete (el máximo para Linja) y a partir de aquí, la acumulación de fichas en determinadas casillas y la ausencia de fichas del jugador en algunas reduce los posibles movimientos y con ello el factor de ramificación.

Analizando el gráfico desde el punto de vista de las diferentes profundidades del algoritmo comprobamos que para un valor de 3 y 4 el coste es prácticamente irrelevante, para 5 se puede considerar aceptable, pero a partir de una profundidad de valor 6 los primeros turnos de una partida pueden resultar desesperantes.



Tiempos de ejecución (ms) del algoritmo Minimax para profundidades 3 a 6

6.2.2 Función heurística de evaluación

Como se ha comentado en el capítulo anterior, la función heurística de evaluación es la encargada de determinar qué movimientos son mejores de otros. Así pues, para que el algoritmo resulte efectivo es fundamental que esta función refleje de la forma más aproximada posible la valoración real de cada estado del tablero. La existencia de varias estrategias posibles hace que esto resulte complejo y usualmente sólo gracias a la experimentación podemos determinar qué funciones de evaluación son mejores que otras.

La función de evaluación obtiene la valoración del estado del tablero a partir de una o diversas funciones de puntuación. Cada función de puntuación determina su valor a partir de una estrategia específica. Por ejemplo, una estrategia podría ser impedir que el oponente obtenga turnos extra y otra podría ser maximizar la puntuación del jugador (avanzando sus fichas todo

cuanto sea posible). Cada una de estas dos estrategias da lugar a su respectiva función de puntuación y la suma ponderada de estas a la función de evaluación. Según la importancia que les otorguemos a cada estrategia podemos variar el peso que cada función de puntuación tiene en el conjunto y primar unas estrategias sobre otras.

Funciones de puntuación para Linja

Antes de comenzar a crear funciones de puntuación que evalúen los posibles estados del tablero del juego Linja conviene detenerse a pensar en las posibles estrategias que un jugador humano podría adoptar.

En Linja obtiene la victoria el jugador que más puntos suma al final de la partida y la única manera que tiene un jugador de conseguir puntos es avanzar sus fichas todo cuanto le sea posible. Por este motivo es natural que la primera estrategia que se le ocurra a un jugador sea la de primar el avance de sus fichas por encima de todo. Basándonos en esto se han ideado ciertos *criterios de avance* que dan lugar a sus respectivas funciones de puntuación.

Pero un jugador que se centre sólo en ampliar su puntuación puede estar infravalorando las opciones de movimiento que ofrece inconscientemente a su adversario. ¿De qué sirve alcanzar con una de mis fichas la casilla del extremo opuesto del tablero si con ello estoy dejando vacía una casilla intermedia que puede aprovechar mi oponente para conseguir un turno extra? ¿Por qué debería avanzar una ficha a determinada casilla que ya cuenta con cuatro de ellas, ofreciéndole al contrario cinco puntos de movimiento? Teniendo en cuenta estas y otras preguntas que un jugador puede hacerse se han pensado unos *criterios de control de territorio*, que priman distintas disposiciones de las fichas en las casillas centrales por encima de otras.

Combinando estos criterios y ponderándolos con distintos pesos en la función heurística de evaluación obtendremos diferentes *personalidades* de la inteligencia artificial: jugadores más

agresivos, otros más cautos o incluso otros más irresponsables. Solamente la experimentación nos responderá a la pregunta de cuáles son mejores que otros y de si hay una estrategia (o conjunto de ellas) que destaca sobre las demás.

Criterios de avance

Estos criterios evalúan el estado del tablero basándose en la puntuación de las fichas según la distancia a las casillas de los extremos.

1. *Puntuación positiva*
    - Criterio que puntúa el estado del tablero como si la partida hubiera acabado.
    - La casilla del extremo opuesto otorga 5 puntos por ficha y las siguientes 3, 2 y 1.
    - Fuerza a avanzar las fichas, especialmente a la casilla del extremo opuesto.

2. *Puntuación negativa*
    - Criterio que puntúa negativamente tener atrasadas las fichas del jugador.
    - La casilla de inicio otorga -5 puntos por ficha y las siguientes -3, -2 y -1.
    - Fuerza a avanzar las fichas, especialmente las situadas en la casilla de inicio.
    - Aumenta el número de fichas propias en las casillas intermedias del tablero.

3. *Puntuación positiva oponente*
    - Criterio que puntúa negativamente según la puntuación actual del oponente si la partida acabara en este momento.
    - Las fichas del oponente en nuestra casilla de inicio otorgan -5 puntos y en las siguientes -3, -2 y -1.
    - Prioriza impedir que nuestro adversario alcance nuestra mitad del tablero.

4. *Puntuación negativa oponente*

    - Criterio que puntúa positivamente que el oponente tenga atrasadas las fichas
    - Las fichas del oponente en su casilla de inicio otorgan 5 puntos y en las siguientes 3, 2 y 1.
    - Prioriza impedir que nuestro adversario avance las fichas en su mitad del tablero.

Criterios de control de territorio

Estos criterios evalúan el estado del tablero según la distribución de las fichas en las casillas, teniendo en cuenta los puntos de movimiento o turnos extra que se le pueden estar ofreciendo al oponente, así como los que podemos obtener nosotros.

5. *Casilla vacía*

    - Criterio que valora la existencia de casillas vacías en el tablero
    - Puntúa negativamente la cantidad de casillas vacías si el oponente jugará en el turno siguiente
    - Puntúa positivamente la cantidad de casillas vacías si es el propio jugador quien jugará el turno siguiente
    - Evita que el oponente pueda obtener un turno extra desplazando en su segundo movimiento una ficha a una casilla vacía; mientras que premia que el propio usuario pueda hacerlo

6. *Casillas completas*

    - Criterio que valora la existencia de casillas intermedias completas (con 6 fichas)
    - Mientras que una casilla con 5 fichas otorga 5 puntos de movimiento al jugador que desplace una ficha a ella en su segundo movimiento; a una casilla completa no

se pueden desplazar fichas y por lo tanto no otorga ninguna ventaja, además de ser un obstáculo.

- Puntúa positivamente que haya casillas completas cuando en el siguiente turno jugará el oponente
- Puntúa negativamente que haya casillas completas cuando es el propio jugador quien moverá en el próximo turno

7. Distribución uniforme

- Criterio que valora una distribución uniforme de las fichas en las casillas
- Evita que el tablero esté descompensado, con algunas casillas con pocas fichas y otras con muchas; y por lo tanto que se puedan obtener un número alto de puntos de movimiento

6.2.3 Definición y evaluación de oponentes IA

Iteración primera

A partir de los criterios tácticos especificados en el capítulo anterior se han creado varios oponentes de IA. En esta primera iteración cada uno estos jugadores virtuales seguirá un solo **criterio táctico de avance**, por lo que la función de evaluación solamente se basará en una sola función de puntuación.

Estos son los jugadores IA iniciales y el criterio que siguen. Para facilitar su reconocimiento se les ha dado nombres de persona:

- David: Puntuación positiva (100%)
- Raúl: Puntuación negativa (100%)
- Jordi: Puntuación positiva oponente (100%)
- Roberto: Puntuación negativa oponente (100%)

Tras enfrentar a estos cuatro oponentes IA entre sí, a continuación se muestran los resultados obtenidos. Conviene recordar que ante la ausencia de azar en el juego Linja, un enfrentamiento entre dos jugadores virtuales arrojará el mismo resultado una y otra vez (pues siempre escogerán los mismos movimientos).

		Jugador negro			
Jugador rojo		David	Raúl	Jordi	Roberto
	David	50/60	56/38	55/54	60/30
	Raúl	38/56	37/35	35/46	60/42
	Jordi	52/58	51/36	45/45	60/30
	Roberto	31/60	41/60	30/60	60/44

En esta tabla podemos comprobar que el jugador David ha ganado a todos los demás (salvo cuando juega contra sí mismo, que curiosamente gana la versión negra, que comienza segunda). Recordemos que David es la IA que prioriza avanzar sus fichas en todo momento (busca maximizar la puntuación).

Contrariamente el jugador virtual que peor se desempeña es Roberto, que toscamente intenta priorizar los movimientos que retrasan las fichas de su oponente.

El segundo peor jugador sería Raúl, quien procura por todos los medios no tener fichas retrasadas en el tablero, y por consiguiente saca enseguida todas ellas de su casilla de inicio.

De esta primera ronda de simulaciones podemos aprender lo siguiente:

- Ante la ausencia de estrategias más complejas, el jugador que priorice maximizar la puntuación será el que gane.
- Desplazar cierta cantidad de fichas al centro del tablero puede resultar perjudicial, posiblemente porque se le está ofreciendo al oponente más puntos de movimiento.

Iteración segunda

Tras los resultados observados se ha decidido mantener al jugador David e introducir nuevos jugadores con más de un **criterio táctico de avance**:

- David: Puntuación positiva (100%)
- Miquel: Puntuación positiva (50%) + Puntuación negativa (50%)
- Carlos: Puntuación positiva oponente (50%) + Puntuación negativa oponente (50%)

El jugador Miquel intentará mantener un equilibrio entre avanzar las fichas y no tenerlas muy retrasadas; mientras que Carlos hará lo mismo pero con las fichas de su oponente (negativamente).

Los resultados de la segunda ronda de simulaciones es el siguiente:

		Jugador negro		
		David	Miquel	Carlos
Jugador rojo	David	-	56/46	60/50
	Miquel	58/52	50/40	60/38
	Carlos	40/60	42/60	45/60

Podemos observar que el jugador Carlos ha perdido todas las partidas (salvo la que jugó contra él mismo, en la cual ganó el color negro). Entre los jugadores David y Miquel vemos que cada uno ha ganado al otro cuando ha sido el primer jugador en mover.

Lo que podemos extraer de estos resultados es que definitivamente, intentar frenar el avance del oponente en base a la puntuación del tablero en cada momento es inútil. Habrá que buscar estrategias más complejas para ello.

Iteración tercera

Para esta tercera simulación se ha decidido añadir oponentes con los **criterios de control de territorio** y enfrentarlos contra los dos jugadores más competentes hasta ahora, David y Miquel.

- David: Puntuación positiva (100%)
- Miquel: Puntuación positiva (50%) + Puntuación negativa (50%)
- Iván: Disposición uniforme (100%)
- Joan: Casillas completas (100%)
- Fran: Casillas vacías (100%)

Así pues, las partidas de la tercera ronda se desarrollaron así:

		Jugador negro				
		David	Miquel	Iván	Joan	Fran
Jugador rojo	David	-	-	55/40	58/44	55/58
	Miquel	-	-	54/22	38/22	46/36
	Iván	30/60	36/60	46/60	51/60	45/60
	Joan	35/60	36/60	51/60	56/48	50/60
	Fran	36/58	51/58	45/46	44/56	54/43

Los nuevos oponentes no se han desenvuelto muy bien contra los dos veteranos. Es comprensible porque sus criterios tácticos evalúan condiciones concretas del tablero (completar una casilla o no dejar casillas vacías) y se olvidan del objetivo primordial del juego, que es avanzar las fichas más que el rival.

Con esto en mente, para la siguiente iteración se mezclarán los criterios de avance con los de control de territorio.

Iteración cuarta

- David: Puntuación positiva (100%)
- Miquel: Puntuación positiva (50%) + Puntuación negativa (50%)
- Tomás: Puntuación positiva (20%) + Puntuación negativa (20%) + Disposición uniforme (20%) + Casillas completas (20%) + Casillas vacías (20%)
- Alberto: Puntuación positiva (40%) + Disposición uniforme (20%) + Casillas completas (20%) + Casillas vacías (20%)
- Rebeca: Puntuación positiva (50%) + Disposición uniforme (16,6%) + Casillas completas (16,6%) + Casillas vacías (16,6%)

Los tres nuevos jugadores combinan varios criterios, algunos idénticos pero con distinto peso. Se intentará determinar un equilibrio entre la estrategia de avanzar las fichas y las de bloquear casillas e impedir que el oponente pueda sacar beneficio de las casillas vacías para obtener un turno extra.

		Jugador negro				
Jugador rojo		David	Miquel	Tomás	Alberto	Rebeca
	David	-	-	55/47	51/60	51/60
	Miquel	-	-	44/28	50/52	50/52
	Tomás	45/54	47/48	42/41	58/47	58/47
	Alberto	58/52	50/58	51/58	53/45	52/47
	Rebeca	58/52	50/58	51/57	53/45	52/47

Esta tabla de resultados arroja alguna información interesante:

- Quizá lo que salte antes a la vista es que dos jugadores (Alberto y Rebeca) obtienen puntuaciones calcadas (¡salvo en una partida!). Esto es así porque usan las mismas funciones de puntuación y la diferencia de peso no hace variar el comportamiento.
- Pero lo más importante: las IA's de David y Miquel pierden contra Alberto y Rebeca (especialmente David, no tanto Miquel). Esto supone un alivio, porque podemos comprobar que la simple estrategia de limitarse a avanzar fichas no *rompe* el juego.
- Además, el jugador Tomás vence a Alberto y Rebeca pero pierde ante David y Miquel. Estas victorias cíclicas (como el juego de piedra, papel y tijera) lo vuelve todo más interesante.

Iteración quinta

Después de observar que hay varias IAs que se ganan entre sí cíclicamente, se ha decidido mantenerlas y añadir nuevos jugadores en los que se variará el peso de los tres criterios de control de territorio, teniendo como base la IA de Miquel.

- David: Puntuación positiva (100%)
- Miquel: Puntuación positiva (50%) + Puntuación negativa (50%)
- Tomás: Puntuación positiva (20%) + Puntuación negativa (20%) + Disposición uniforme (20%) + Casillas completas (20%) + Casillas vacías (20%)
- Alberto: Puntuación positiva (40%) + Disposición uniforme (20%) + Casillas completas (20%) + Casillas vacías (20%)
- Mónica: Puntuación positiva (14,3%) + Puntuación negativa (14,3%) + **Disposición uniforme (42,85%)** + Casillas completas (14,3%) + Casillas vacías (14,3%)
- Lucía: Puntuación positiva (14,3%) + Puntuación negativa (14,3%) + Disposición uniforme (14,3%) + **Casillas completas (42,85%)** + Casillas vacías (14,3%)
- Óscar: Puntuación positiva (14,3%) + Puntuación negativa (14,3%) + Disposición uniforme (14,3%) + Casillas completas (14,3%) + **Casillas vacías (42,85%)**

Cada uno de los tres nuevos jugadores pone énfasis en uno de los criterios de control de territorio, mientras mantiene el resto de criterios con el mismo peso. Esto podría ayudarnos a determinar cuál de los tres resulta más decisivo. A priori se podría pensar que el criterio que tiene en cuenta las casillas vacías (dando pie a que los jugadores obtengan turnos extra o no) es más importante que el que se preocupa de completar las casillas. Al fin y al cabo, un turno extra pinta mejor que desplazar cuatro posiciones. Comprobémoslo.

		Jugador negro						
		David	Miquel	Tomás	Alberto	Mónica	Lucía	Óscar
Jugador rojo	David	-	-	-	-	53/60	48/60	48/60
	Miquel	-	-	-	-	55/45	56/43	52/46
	Tomás	-	-	-	-	47/40	42/41	58/55
	Alberto	-	-	-	-	53/47	50/48	45/55
	Mónica	45/54	48/58	45/49	55/51	54/50	45/49	48/43
	Lucía	49/55	37/41	42/50	53/55	56/35	47/44	56/47
	Óscar	49/55	43/46	52/44	54/52	51/44	52/43	52/41

Observando la tabla comprobamos que los jugadores nuevos han ganado: Óscar, siete partidas; Lucía, 5 partidas; y Mónica, 4 partidas.

Parece que se cumple la previsión de que el criterio de casillas vacías prevalece sobre los otros. No obstante habría que afinar el balanceo de las funciones de evaluación y encontrar un equilibrio, ya que Óscar todavía sigue perdiendo ante David y Miquel.

Llegado a este punto convendría razonar uno de los comportamientos observados en las IAs a lo largo de las sucesivas iteraciones:

- En ocasiones algunos oponentes IA pierden la partida en los últimos turnos, incluso cuando la han estado dominando claramente en puntuación desde el principio. Esto es así porque mantienen fija su táctica, cuando probablemente les resultara más beneficioso forzar el final de la partida sobrepasando las fichas de su oponente cuanto antes.

Esto nos lleva a la conclusión que probablemente una única táctica uniforme a lo largo del curso de la partida puede no ser la mejor estrategia. Tal vez convendría que los jugadores IA cambiaran su estilo de juego (alterando el peso de sus funciones de puntuación *in-game*) adaptándolo a las circunstancias de cada fase de la partida. Por ejemplo y como se ha

comentado anteriormente, los jugadores deberían *olvidar* su criterio táctico establecido y centrarse exclusivamente en forzar el final de la partida si ven que éste está próximo y van ganando.

Con el objetivo de modificar el comportamiento de los jugadores a lo largo de la partida se ha ideado este primer nuevo criterio táctico:

8. Opciones de movimiento

- Este criterio está pensado para tomar el control de las decisiones de la IA en la fase final de la partida, cuando apenas quedan fichas y por consiguiente hay menos opciones de movimiento.
- Puntúa el tablero en función del número de movimientos distintos entre los que podrá elegir el siguiente jugador.
- Si el próximo jugador es el rival, maximiza las situaciones en las que éste no tiene opciones de movimiento (y por lo tanto tiene que pasar turno).
- Si el próximo turno es el de nuestro jugador, minimiza las situaciones en las que no puede mover.
- La valoración es exponencial. Normalmente a lo largo de la partida los jugadores tendrán en torno a seis o siete movimientos para elegir; en estos casos resulta irrelevante este criterio táctico, pero puede ser altamente decisivo a medida que se reducen las opciones.

En la sexta iteración se introducirán dos nuevos jugadores que seguirán esta táctica.

Iteración sexta

Los nuevos jugadores que tienen en cuenta el nuevo criterio de Opciones de movimiento son:

- Tania: Puntuación positiva (25%) + Puntuación negativa (25%) + Opciones de movimiento (50%)
- Daniel: Puntuación positiva (7,14%) + Puntuación negativa (7,14%) + Disposición uniforme (7,14%) + Casillas completas (7,14%) + Casillas vacías (21,4%) + Opciones de movimiento (50%)

Se han creado en base a los antiguos jugadores Miquel y Óscar respectivamente.

Esta nueva ronda de enfrentamientos ha aportado la siguiente información:

		Jugador negro								
Jugador rojo		David	Miquel	Tomás	Alberto	Mónica	Lucía	Óscar	Daniel	Tania
	David	-	-	-	-	-	-	-	55/55	48/60
	Miquel	-	-	-	-	-	-	-	56/41	58/53
	Tomás	-	-	-	-	-	-	-	54/44	56/53
	Alberto	-	-	-	-	-	-	-	50/58	50/47
	Mónica	-	-	-	-	-	-	-	49/48	49/55
	Lucía	-	-	-	-	-	-	-	58/44	49/51
	Óscar	-	-	-	-	-	-	-	58/41	49/51
	Daniel	51/56	53/58	46/50	51/52	57/47	43/53	43/53	55/55	52/46
	Tania	57/55	50/50	47/45	55/46	45/42	50/45	52/50	55/50	51/45

Lo cierto es que el desempeño del jugador Daniel ha sido bastante decepcionante, pero afortunadamente no se puede decir lo mismo de Tania, quien ha ganado once partidas, empatado una y perdido cuatro (sin contar la jugada contra sí misma).

Estos resultados nos indican que se está en la buena dirección: hay que desarrollar más criterios que evalúen el tablero en función del momento de la partida.

6.2.4 Conclusiones del análisis de la estrategia de Linja

La verdad es que el análisis y desarrollo de tácticas que implementar y con las que experimentar ofrece un amplio abanico de posibilidades, a las que lamentablemente no se les puede dedicar todos los recursos destinados a este proyecto.

A pesar de ello, las numerosas pruebas y simulaciones que se han llevado a cabo permiten extraer algunas conclusiones y hacernos una idea de hacia dónde se debe avanzar y profundizar en el desarrollo de una IA para Linja. Lo que podemos destacar es lo siguiente:

- No hay ninguna táctica que mantenida invariable a lo largo de la partida otorgue con seguridad la victoria. No hay una táctica ganadora sobre el resto. O por lo menos no se ha encontrado.
- Cualquier nueva IA que se desarrolle debe saber adaptarse al momento actual de la partida, especialmente cuando se acerca el final de la misma. Debe detectar cuándo forzar el fin del juego, avanzando sus fichas más atrasadas para sobrepasar a su oponente (en el caso de que vaya ganando), sin preocuparle el tener que abandonar su táctica inicial.
- Una buena combinación de táctica inicial que permita al jugador IA adelantarse en puntos a su adversario a media partida y una táctica de forzar el final podría ser una excelente opción para una nueva iteración en las simulaciones.

6.2.5 Propuesta de mejora del algoritmo IA

Además de inventar nuevos criterios tácticos, como se ha explicado en el apartado anterior, también se podría considerar realizar una mejora del algoritmo IA para rebajar el coste de ejecución.

Para una futurible segunda fase del proyecto se podría trabajar en aumentar la profundidad del algoritmo Minimax. A mayor número de turnos vista que se juegue, mejor se podrá afinar la táctica utilizada. Desgraciadamente un aumento de la profundidad implica un mayor tiempo de ejecución.

Si tenemos en mente el árbol de juego, con todas las posibles decisiones que se pueden tomar en la partida, podemos imaginar que movimientos actuales que tienen efectos distintos pueden acabar llegando a obtener -a varios turnos vista- la misma distribución de fichas sobre el tablero. Cuando el algoritmo Minimax que usamos realiza el recorrido del árbol de juego probablemente esté evaluando también algunos estados del tablero que ha podido evaluar anteriormente. Es decir, está calculando dos o más veces lo mismo.

Para paliar esto se podría utilizar un algoritmo MTD. Este tipo de algoritmos se caracterizan por usar una tabla de transposiciones: una tabla hash donde se almacenan los estados previamente buscados. Así pues, en una nueva búsqueda y antes de continuar el recorrido del árbol, el algoritmo buscará si existe dicho estado en la tabla de transposición. Si lo encuentra, tomará el resultado y ahorrará tiempo de ejecución. Si no lo encuentra, evaluará la sección del árbol de juego como de costumbre y lo añadirá a la tabla hash para ahorrar en futuras búsquedas.

Esta podría tratarse de una excelente solución para lograr jugar a mayores profundidades del algoritmo Minimax y daría pie a estudiar el comportamiento de los criterios tácticos a diferentes profundidades.

7. Planificación estratégica y coste económico

7.1 Planificación inicial

Al comienzo de este proyecto se llevó a cabo una estimación general sobre la planificación estratégica del mismo.

En la definición de la planificación se tuvo en cuenta fundamentalmente el número de horas aproximado que se evalúa debe durar un proyecto de estas características (22,5 créditos para Ingeniería en Informática de Gestión, a razón de 20 horas por crédito unas 400-450 horas) para realizar el reparto de tiempo entre las distintas tareas.

La estimación que se hizo fue la siguiente:

Fase inicial	20
Especificación	20
Diseño	30
Estudio de la tecnología	50
Implementación	120
Implementación modo local	*70*
Implementación modo multijugador	*50*
Diseño e implementación de la IA	100
Redacción de la memoria	60
Total	**400**

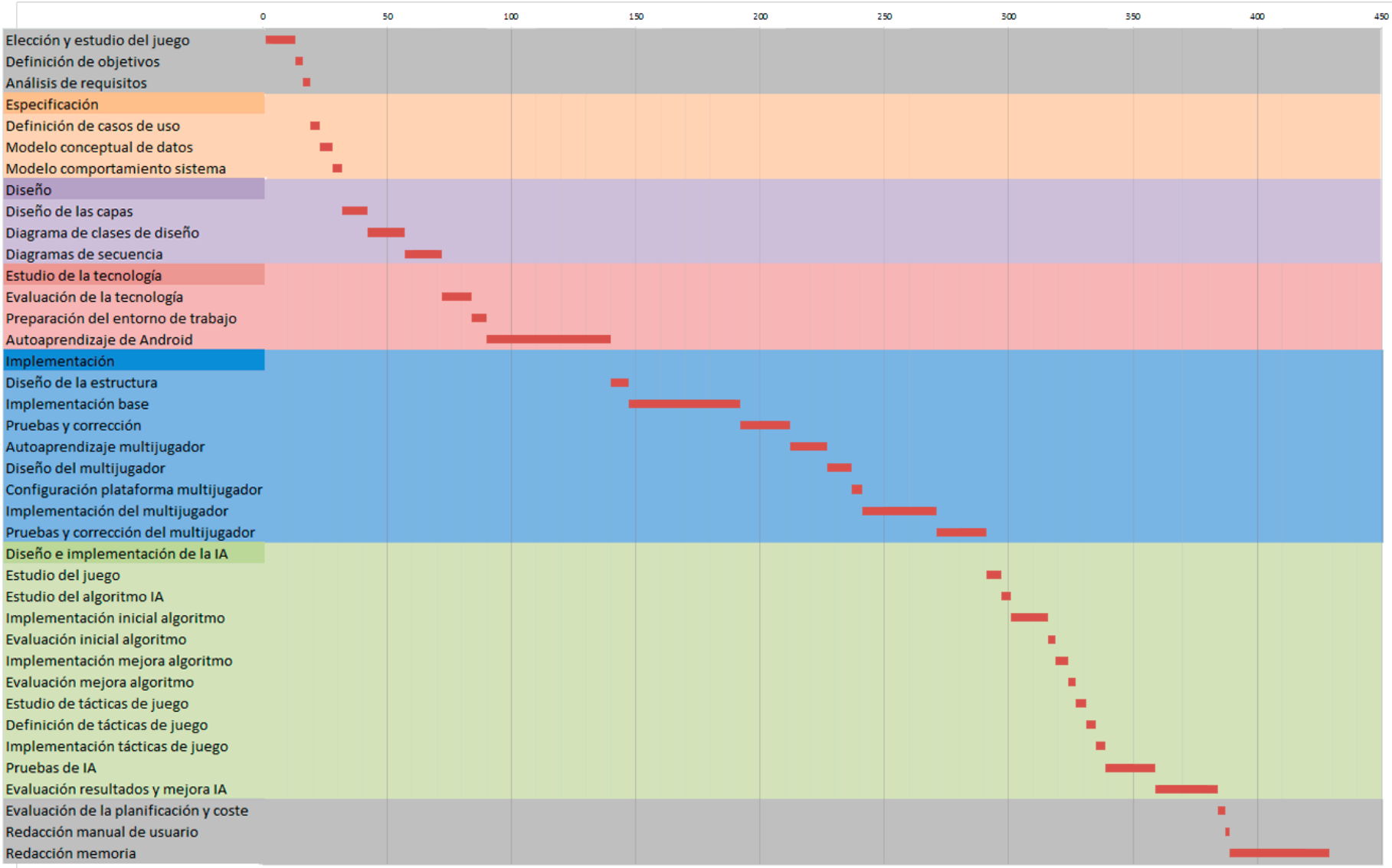
Se decidió ajustar al máximo el número de horas planificado para cada tarea de forma que permitiese dejar cierto margen para cubrir cualquier imprevisto.

7.2 Planificación final

Finalmente, el cómputo total de horas desglosadas específicamente para cada subtarea ha sido el siguiente:

Fase inicial	
Elección y estudio del juego	12
Definición de objetivos	3
Análisis de requisitos	3
Total	**18**
Especificación	
Definición de casos de uso	4
Modelo conceptual de datos	5
Modelo comportamiento sistema	4
Total	**13**
Diseño	
Diseño de las capas	10
Diagrama de clases de diseño	15
Diagramas de secuencia	15
Total	**40**
Estudio de la tecnología	
Evaluación de la tecnología	12
Preparación del entorno de trabajo	6
Autoaprendizaje de Android	50
Total	**68**

Implementación	
Diseño de la estructura	7
Implementación base	45
Pruebas y corrección	20
Autoaprendizaje multijugador	15
Diseño del multijugador	10
Configuración plataforma multijugador	4
Implementación del multijugador	30
Pruebas y corrección del multijugador	20
Total	**151**
Diseño e implementación de la IA	
Estudio del juego	6
Estudio del algoritmo IA	4
Implementación inicial algoritmo	15
Evaluación inicial algoritmo	3
Implementación mejora algoritmo	5
Evaluación mejora algoritmo	3
Estudio de tácticas de juego	4
Definición de tácticas de juego	4
Implementación tácticas de juego	4
Pruebas de IA	20
Evaluación resultados y mejora IA	25
Total	**93**
Fase final	
Evaluación de la planificación y coste	3
Redacción manual de usuario	2
Redacción memoria	40
Total	**45**
Total general	**428**


0
50
100
150
200
250
300
350
400
450
Elección y estudio del juego
Definición de objetivos
Análisis de requisitos
Especificación
Definición de casos de uso
Modelo conceptual de datos
Modelo comportamiento sistema
Diseño
Diseño de las capas
Diagrama de clases de diseño
Diagramas de secuencia
Estudio de la tecnología
Evaluación de la tecnología
Preparación del entorno de trabajo
Autoaprendizaje de Android
Implementación
Diseño de la estructura
Implementación base
Pruebas y corrección
Autoaprendizaje multijugador
Diseño del multijugador
Configuración plataforma multijugador
Implementación del multijugador
Pruebas y corrección del multijugador
Diseño e implementación de la IA
Estudio del juego
Estudio del algoritmo IA
Implementación inicial algoritmo
Evaluación inicial algoritmo
Implementación mejora algoritmo
Evaluación mejora algoritmo
Estudio de tácticas de juego
Definición de tácticas de juego
Implementación tácticas de juego
Pruebas de IA
Evaluación resultados y mejora IA
Evaluación de la planificación y coste
Redacción manual de usuario
Redacción memoria

7.3 Comparativa de la planificación

Observando la planificación estimada con la real comprobamos que se ha producido un desfase de 28 horas. Analicemos el cómputo en horas de cada tarea para obtener resultados más reveladores:

	Estimación	Real	Diferencia
Fase inicial	20	18	**-2**
Especificación	20	13	**-7**
Diseño	30	40	**10**
Estudio de la tecnología	50	68	**18**
Implementación	120	151	**31**
Implementación modo local	*70*	*72*	***2***
Implementación modo multijugador	*50*	*79*	***29***
Diseño e implementación de la IA	100	93	**-7**
Redacción de la memoria	60	45	**-15**
Total	**400**	**428**	**28**

El desfase más importante se ha producido en la implementación, concretamente en la del modo multijugador. La causa de esto ha sido la aparición de algunos problemas imposibles de prever en la planificación inicial, aunque se podría haber contemplado distribuir más horas hacia esta tarea debido a la ausencia de conocimientos sobre el desarrollo multijugador.

A pesar de que en la comparativa pasa desapercibido, hay que señalar que la parte fundamental del diseño y la implementación de la Inteligencia Artificial se ha realizado en un tiempo significativamente menor al estimado, lo cual ha permitido derivar recursos a ampliar el tiempo dedicado a las pruebas y la evaluación de las IAs.

7.4 Coste económico

Una vez obtenido el cálculo en horas de cada tarea del proyecto podemos proceder a calcular el coste económico del mismo.

Desde el punto de vista de **recursos humanos** se han diferenciado tres perfiles:

- Analista
- Programador
- Probador de software (tester)

Tras distribuir las tareas realizadas (salvo la redacción de la memoria) entre los tres perfiles y asignarles una retribución a cada uno de ellos, obtenemos el siguiente coste:

Perfil	Horas	€/hora	Coste
Analista	126	45	5.670,00 €
Programador	222	30	6.660,00 €
Tester	40	15	600,00 €
			12.930,00 €

El resto del coste corresponde a los materiales, hardware, software o licencias. La mayoría de los equipos hardware estaban amortizados, por lo que se han omitido.

Material	Precio
Terminal Motorola G	152,00 €
Licencia Desarrollador Google	23,35 €
Juego de mesa Linja	12,60 €
	187,95 €

Así pues, el coste total del proyecto asciende a **13.117,95€.**

8. Conclusiones

El objetivo de este proyecto de final de carrera era el desarrollo para dispositivos móviles de una adaptación del juego de mesa Linja que incorporase diversas funcionalidades, como permitir a dos usuarios jugar una partida local en el mismo dispositivo, enfrentarse un jugador contra un oponente de Inteligencia Artificial u ofrecer la posibilidad al usuario de poder jugar contra otros adversarios a través de Internet.

Todos estos objetivos han sido cubiertos. La aplicación funciona correctamente y el usuario puede jugar contra IAs y contra usuarios de Internet.

Pero en un proyecto de este tipo siempre hay cosas que se pueden mejorar y otras interesantes que se descubren en el camino y aportan nuevas ideas sobre qué hacer, o más bien sobre qué hacer si se dispusiera de más tiempo para ello.

Me refiero especialmente al aspecto del desarrollo de la Inteligencia Artificial. Linja es un juego muy sencillo, no hay que negarlo. No hay azar y sus componentes se limitan a unas simples fichas que se desplazan sobre un tablero en una sola dirección. Pero las posibilidades que aporta elaborar una Inteligencia Artificial son muy amplias, y mejorar y perfeccionar esta Inteligencia puede llegar a suponer un auténtico reto.

En el capítulo de Objetivos de este proyecto definía desarrollar una IA que se “desenvolviera con cierta competencia” contra el usuario. Este concepto puede ser un poco ambiguo, pero lamentablemente el jugador humano no va a encontrar aquí una IA que le suponga un verdadero desafío ni le provoque grandes quebraderos de cabeza el derrotarla.

La creación de la Inteligencia Artificial es lo que más he disfrutado, y aun así todavía se podría trabajar mucho más en ella. En más de una dimensión: desde mejorar el algoritmo Minimax

para reducir su coste de ejecución hasta idear nuevas tácticas para los oponentes virtuales. Y hacer pruebas, muchas pruebas.

Aunque el mundo de los juegos de mesa no me era desconocido, el de la Inteligencia Artificial sí lo era. Este proyecto me ha permitido conocerlo mejor y despertarme la curiosidad sobre cómo implementar una IA para tal o cuál juego. Creo que la próxima vez que juegue a uno inevitablemente no podré evitar analizarlo desde esta nueva perspectiva.

El multijugador es otro aspecto que se podría ampliar. En su estado actual cumple su función y punto. Pero en una segunda versión de la aplicación se podrían explotar las posibilidades que ofrece *Google Play Game Services* y hacer que el juego resultara más atractivo de jugar.

Aunque desarrollar utilizando esta API hace que la parte online de la aplicación Linja sea totalmente dependiente de los servicios de Google, me ha permitido liberar recursos para dedicarlos a otras etapas del proyecto.

Una vez sufragado el coste de registrarse como desarrollador en Google Play, la aplicación podría publicarse con apenas pulsar un botón. Con más tiempo se podría haber lanzado, evaluado su resultado e incluido un capítulo sobre ello en la memoria.

9. Propuesta de mejoras

Durante el proceso de estudio del juego y el análisis de lo que ofrecían las tecnologías utilizadas surgieron diferentes ideas y funcionalidades que por uno u otro motivo no se han podido llegar a materializar (generalmente a causa del factor tiempo).

Aquí se listan algunas de ellas, proponiéndolas para una futurible segunda versión de la aplicación:

- **Mejora del algoritmo de IA.** Principalmente implementar un nuevo algoritmo que reduzca el tiempo de cálculo de movimientos de la IA y le permita jugar a un mayor número de turnos vista (es decir, aumentar la profundidad del algoritmo Minimax).
- **Añadir cierta incertidumbre a la IA.** Actualmente, ante dos movimientos que conducen al mismo valor de evaluación del tablero, la Inteligencia Artificial escoge siempre el mismo (el primero que recibe). Se podría implementar que ante dos movimientos a priori iguales, la IA escogiera uno de ellos al azar. Esto variaría el resultado del enfrentamiento entre dos IAs y podría descolocar a un jugador humano habituado a jugar contra ellas.
- **Implementar logros.** Es algo que está muy de moda en los juegos actuales y motiva a los jugadores a jugar exclusivamente para desbloquearlos. Se podrían añadir tales como: ganar 10, 25, 50, 100 partidas; ganar 5 veces seguidas, vencer a todas las IAs del juego, etc.
- **Implementar rankings online.** Los jugadores escalarían posiciones tras obtener puntos por sus victorias, y por la diferencia de puntos con su oponente. También se podrían comparar en rankings internos con sus amigos, o rankings específicos por país, región, ciudad, etc.

- **Añadir avatares de jugador.** Es una mera floritura pero aportaría vistosidad a los rankings.
- **Añadir personalización de las partidas.** A pesar de la simpleza de Linja, se podría dejar a elección del usuario jugar con diferentes cambios. Por ejemplo, se podría permitir al jugador escoger la disposición inicial de las fichas en el tablero, otorgando ventaja o desventaja al otro jugador. También se podría ampliar el número de casillas, jugar con un número de fichas distinto, etc. Algún cambio menor podría ser añadir la posibilidad de cambiar el color de las fichas.
- **Implementar las nuevas reglas de Linja 2015.** Recientemente este año el autor de Linja ha publicado unas nuevas reglas. Ahora los jugadores obtienen un turno extra cuando llegan con una ficha a la casilla de salida del contrario con el número exacto de movimientos; pero este movimiento extra sólo le permite desplazar una de sus fichas una casilla hacia delante o hacia atrás. Además, al final de la partida las fichas del jugador que estén demasiado atrasadas le otorgarán puntos negativos. Implementar estas nuevas reglas podría ampliar el estudio de la Inteligencia Artificial y comparar qué criterios tácticos funcionan mejor para un tipo de reglas u otro.

10. Bibliografía

Formato físico

Millington, I. (2009). *Artificial Intelligence for Games*. CRC Press.

LARMAN, C. (2004). *Applying UML and patterns: An Introduction to Object-Oriented Analysis and Design and the Iterative Development.* Prentice Hall.

Formato electrónico

Google Developers. *Play Game Services*. < https://developers.google.com/games/services> [2015]

Google Android. *API Guides*. < https://developer.android.com/guide/index.html> [2015]

Stack Exchange Inc. *Stack Overflow*. <http://stackoverflow.com> [2015]

BoardGameGeek, LLC. *Board Game Geek.* <https://boardgamegeek.com> [2015]

11. Apéndices

A. Herramientas utilizadas

Para la realización de este proyecto se han utilizado diferentes recursos hardware y software. Esta es la relación de todos ellos:

Hardware

- Ordenador personal. Procesador: AMD Athlon 64 X2 Dual Core 5200+ 2.70Ghz. Memoria RAM: 2 GB. Sistema operativo: Windows 7 64 bits.
- Teléfono móvil Sony Ericsson Xperia Neo V. Procesador Qualcomm MSM8255 Snapdragon 1 GHz. Memoria RAM: 512MB. Sistema operativo: Android 2.3.4 (Gingerbread).
- Teléfono móvil Motorola Moto G (2nd Gen). Procesador Qualcomm MSM8226 Snapdragon 400 quad-core 1.2 GHz. Memoria RAM: 1GB. Sistema operativo: Android 5.0.2 (Lollipop).

Software

- Android Studio 1.3.1
- Adobe Photoshop CS6
- ArgoUML
- Microsoft Word 2010
- Microsoft Excel 2010

B. Manual de usuario

Requisitos mínimos:

- Sistema operativo Android 2.3.3 (API 10)

Requisitos necesarios (sólo para modo multijugador):

- Aplicación *Google Play Juegos* instalada

Menú principal

El menú principal de la aplicación ofrece las siguientes opciones:

- **Partida local:** Permite a dos personas jugar una partida en el mismo dispositivo.
- **Partida Vs IA:** Permite a una persona jugar contra un oponente IA.
- **Modo online:** Accede al menú multijugador.
- **Enfrentamiento IAs:** Permite enfrentar dos oponentes virtuales entre sí para comprobar quién es mejor.

Partida Vs. IA

Antes de empezar a jugar contra un oponente virtual, el usuario debe escoger contra qué personalidad IA se quiere enfrentar, en qué nivel de dificultad (muy fácil, fácil, medio o difícil) y cuál de los dos comenzará a jugar el primer turno.

Al elegir la opción de **Enfretamiento IAs** el usuario también podrá elegir la personalidad y la dificultad de ambas IAs.

Modo multijugador

Para poder acceder a este modo es necesario tener instalada la aplicación *Play Juegos* de Google. Si no se encuentra instalada o está desactualizada, se le ofrecerá al usuario la posibilidad de descargarla.

Una vez hecho esto, el usuario debe iniciar sesión en su cuenta de Google+ y tras ello se le ofrecerán las siguientes opciones:

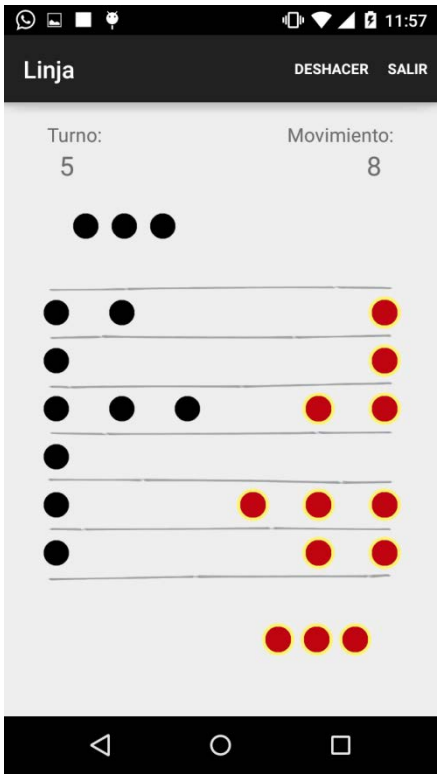
- **Emparejamiento aleatorio:** El sistema buscará un oponente aleatorio al que enfrentarse.
- **Invitar a nueva partida:** El usuario podrá seleccionar de una lista un jugador al que

enviar una invitación para jugar contra él, bien un amigo suyo, un jugador contra el que ha jugado recientemente o un jugador situado geográficamente cerca de él.

- **Ver invitaciones y partidas:** Accede al historial de partidas del usuario, viendo en las que le toca jugar a él y en las que espera el movimiento de otros oponentes, así como las invitaciones a partidas recibidas.
- **Iniciar sesión / cerrar sesión:** Permite conectarse o desconectarse de Google+.

En partida

Para desplazar una ficha solo hay que seleccionarla pulsando sobre ella y seguidamente clicar la casilla a la que se desea mover. Si se quiere deseleccionar una ficha pulsada por error, sólo hay que pulsar el botón *Deshacer* situado en la barra superior de la pantalla.



El botón ***Deshacer*** también sirve para revertir el último movimiento realizado. Si se pulsa repetidas veces revertirá tantos movimientos hasta alcanzar el comienzo de la partida. Si se pulsa tras el turno de una IA, deshará todos los movimientos de la IA y el movimiento previo realizado por el jugador. Este botón se encuentra deshabilitado en el modo multijugador para evitar que oponentes desconocidos hagan mal uso de él.

Para salir de la partida basta con pulsar el botón de ***Salir*** de la barra superior.

C. Sobre el autor de Linja

El alemán Steffen Mülhaüser nació en 1956 y vive en un pequeño pueblo llamado Krastel, situado entre las cuencas de los ríos Rin y Mosela, en el länder germano de Renania-Palatinado. Lleva creando juegos desde el año 1996 y publicándolos con su pequeña editorial desde 2003.

Con más de 20 diseñados a sus espaldas, sus juegos se caracterizan por un estilo sencillo y por utilizar materiales naturales como la madera, lo que transfiere un toque artesano a sus creaciones. Quizá por ello es común verle en ferias de artesanía por toda Alemania, mostrando sus juegos y enseñando a jugar al público curioso.

Linja es probablemente su juego más famoso y conocido, pero entre toda su ludoteca podemos destacar alguno de estos otros títulos:

Schwarzer Kater (Gato negro) es un simpático juego de memoria de 2 a 9 jugadores que recuerda mucho al famoso juego de los trileros. En él varios jugadores hacen de ratones, mientras que un jugador hace de director de juego y oculta varias piezas de madera (que representan dos trozos de queso, una loncha de beicon, una tableta de chocolate y un gato negro) en cinco pequeños cubiletes. Tras unos momentos en el que el

resto de jugadores memorizan la posición de los objetos, el director mueve dos o tres cubiletes e inmediatamente pregunta dónde se encuentra uno de los objetos en cuestión. Los jugadores deben intentar acertar, evitando a toda costa señalar el cubilete del malvado gato. Los ganadores obtienen contadores de victoria que representan trozos de queso y la partida prosigue tres rondas más. Aunque en un principio pueda parecer sencillo lo cierto es que exige ciertas dotes de memorización, ya que el hecho de no saber de antemano el objeto por el que se preguntará lo complica más que si simplemente hubiera que seguir un cubilete con una bolita.

Six es un juego de estrategia para dos jugadores en el que estos se turnan para ir posicionando sus fichas hexagonales en la zona de juego, siempre junto a otra ficha colocada previamente. El objetivo de cada jugador es lograr formar con seis fichas de su color una figura determinada (un círculo, una línea recta o un triángulo; con una sola de ellas es suficiente) mientras impide que su adversario lo consiga antes. Si los jugadores han colocado todas sus fichas sobre la mesa, en una segunda fase de la partida podrán mover y/o forzar la retirada de fichas del tablero, siempre intentando crear una de las tres figuras.

Página web del autor: **http://www.steffen-spiele.de/**